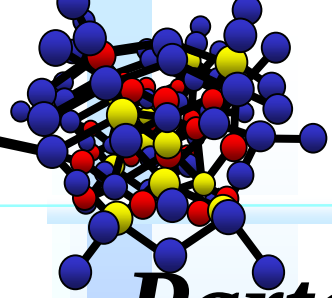


BY NC ND
SOME RIGHTS RESERVED

Parte 6: Architetture Parallele, GRID, big data, MapReduce, Spark

Corso di: Sistemi Distribuiti
Lauree in: Ingegneria Informatica,
delle Telecomunicazioni ed Informatica di Scienze

Prof. Paolo Nesi

Department of Systems and Informatics, University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570

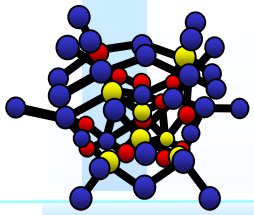
DISIT Lab, Sistemi Distribuiti e Tecnologie Internet

http://www.disit.dinfo.unifi.it/

paolo.nesi@unifi.it
http://www.disit.dinfo.unifi.it/nesi

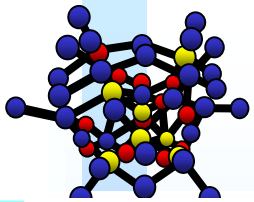
UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO Dipartimento di Ingegneria dell'Informazione

DISIT



sommario

- Contesto tecnologico
- Overhead e Speed-UP
- Aspetti Topologici per il calcolo parallelo
- Soluzioni GRID
- Soluzioni MicroGRID
- Apache Hadoop MapReduce
  - Vedere anche esercitazioni
  - http://www.disit.org/7073
- Apache Hadoop MapReduce 2
- Apache Spark



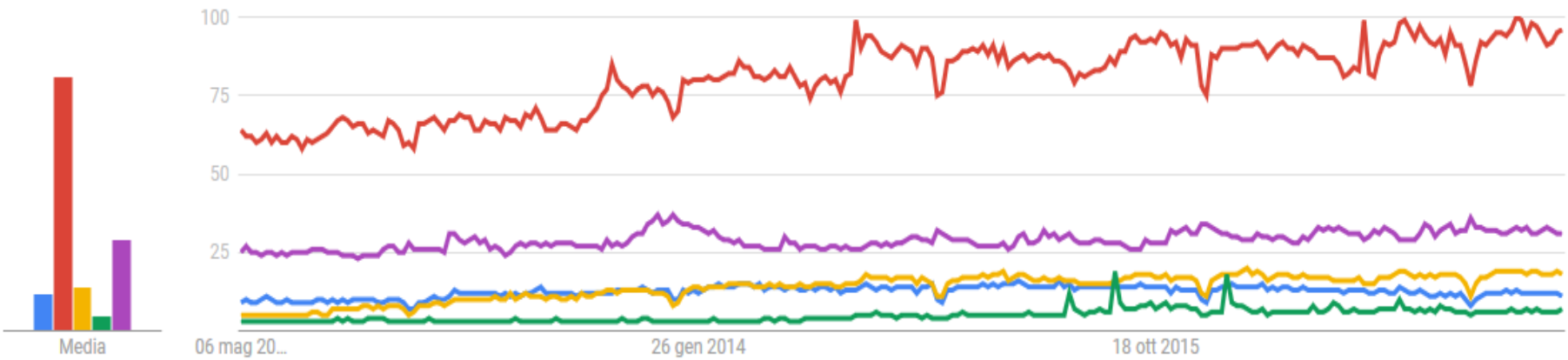
Google Trends

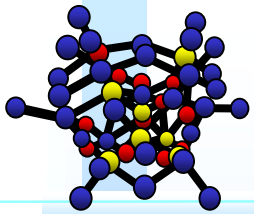
hadoop	Cloud computing	big data	smart city	gpu
Termine di ricerca	Argomento	Termine di ricerca	Termine di ricerca	Termine di ricerca

Tutto il mondo ▾ Ultimi 5 anni ▾ Tutte le categorie ▾ Ricerca Google ▾

I termini di ricerca corrispondono a parole specifiche; gli argomenti sono concetti corrispondenti a termini simili in qualsiasi lingua. Ulteriori informazioni

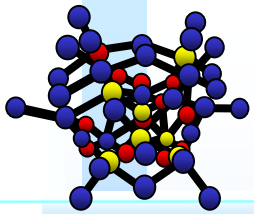
Interesse nel tempo





BigData Science

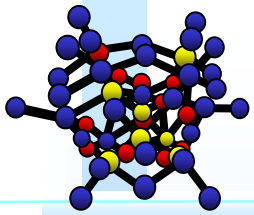
- How to: compute, process, simulate, extract meaning, of vaste/huge amount of data to create knowledge
- Exploit efficiently the huge amount of data/knowledge
- Issues:
  - Data representation: indexing, search, execute, etc..
  - Data computing: sparse, uncertain, fast, etc.
  - Data understanding: mining, analyze, etc.
  - Data view: fast, navigate,
- Applications:
  - Recommendations, suggestions, semantic computing
  - Business intelligence: health, trends, genomic, HBP,
  - Distributed database, decision taking
  - Social networking, smart city
  - Event detection, unexpected correlation



Campi di Utilizzo

- Media distribution
  - Profiling and personalization
  - Produzione di suggerimenti
  - Content delivering network
  - Adattamento di risorse digitali,
  - conversione di formato
- Visione artificiale
  - Riconoscimento di facce, tracciamento
  - Composition/mosaicing of GIS images
  - Riconoscimento delle tracce audio: fingerprint
- Costruzione di contenuti in automatico, NEWS, etc.
  - Generazione di descrittori di risorse digitali
  - Indicizzazione e preparazione alle query
  - Natural Language Processing, affective computing

- *Medicali*
- *Simulazione strutturali, fluidodinamica, deformazioni, finanziarie, servizi, etc.*
- *Predizioni del tempo*
- *Predizioni finaziarie*



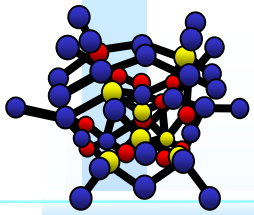
Il Contesto Tecnologico

Crescita delle risorse

- Il numero di transistor raddoppia ogni 18 mesi (Legge di Moore)
- La velocità dei computer raddoppia ogni 18 mesi
- La densità di memoria raddoppia ogni 12 mesi
- La velocità della rete raddoppia ogni 9 mesi

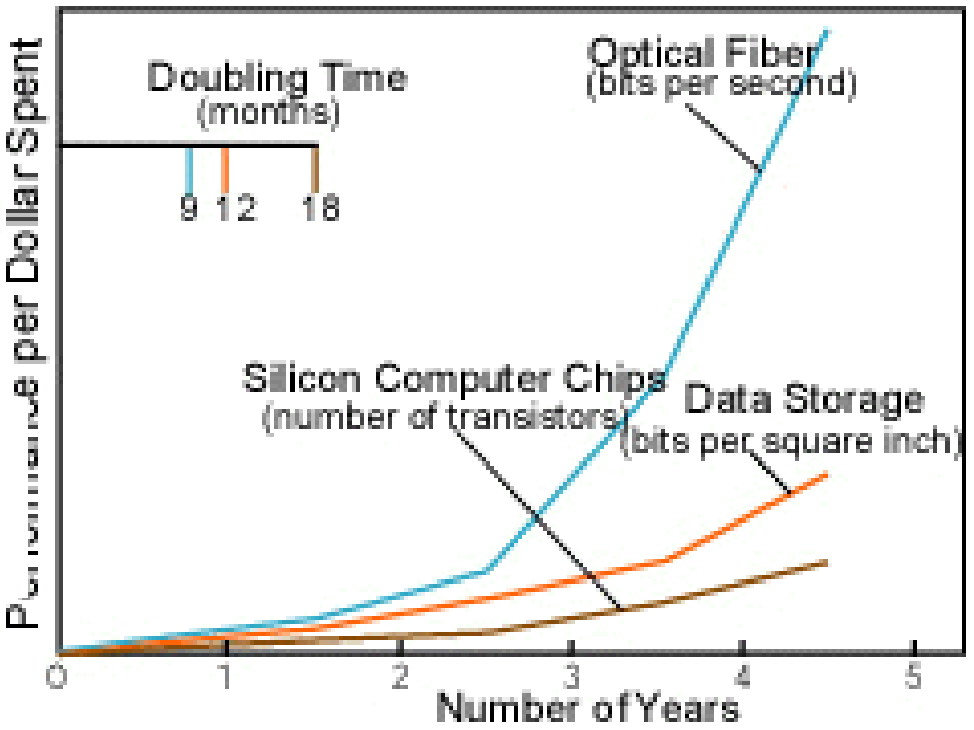
Differenza = un ordine di grandezza ogni 5 anni

Pertanto ogni anno che passa diventa piu' conveniente usare delle soluzioni distribuite.



Network Exponentials

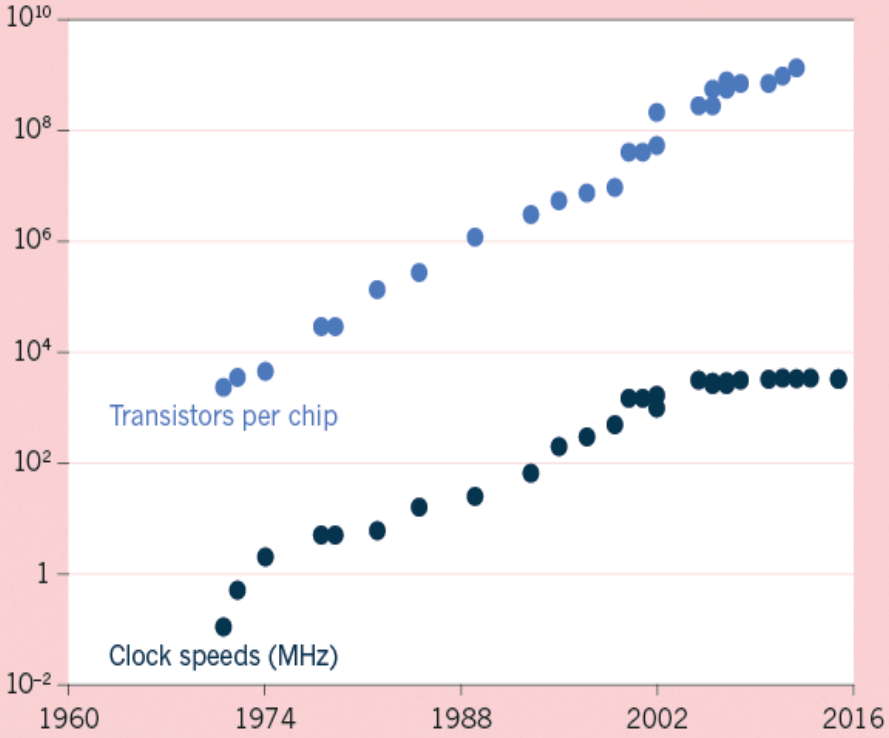
- Network vs. computer performance
  - Computer speed doubles every 18 months
  - Network speed doubles every 9 months
  - Difference = one order of magnitude every 5 years
- 1986 to 2000
  - Computers: x 500
  - Networks: x 340,000
- 2001 to 2010
  - Computers: x 60

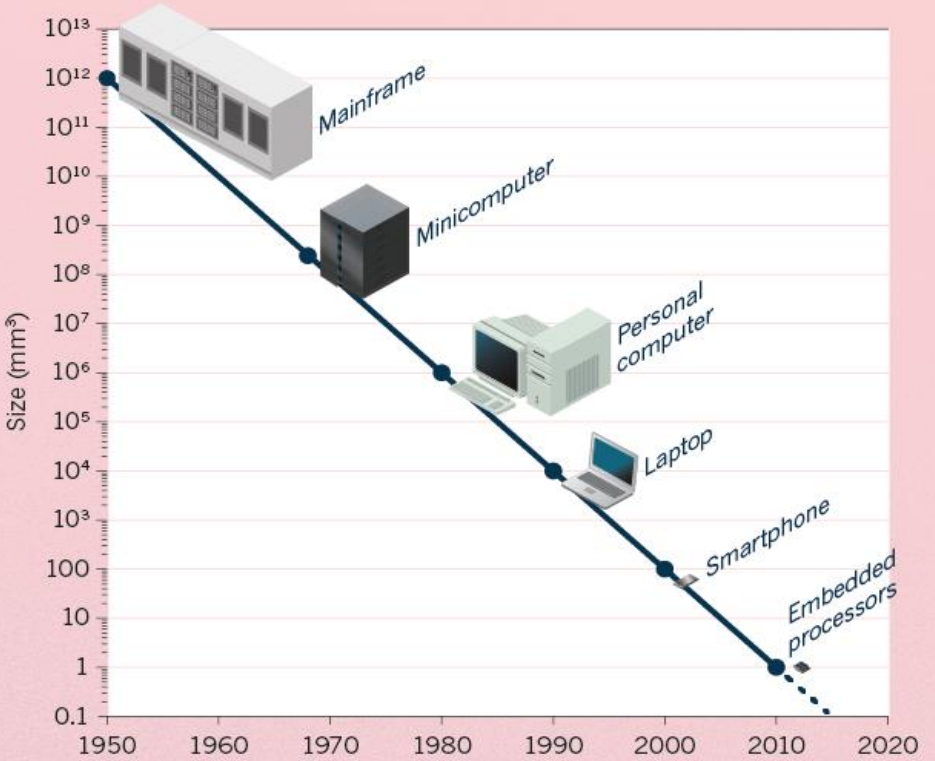


Moore's Law vs. storage improvements vs. optical improvements. *Graph from* ***Scientific American*** *(Jan-2001) by Cleo Vilett, source Vined Khoslan, Kleiner, Caufield and Perkins.*

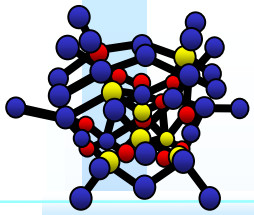
MOORE'S LORE

For the past five decades, the number of transistors per microprocessor chip — a rough measure of processing power — has doubled about every two years, in step with Moore's law (top). Chips also increased their 'clock speed', or rate of executing instructions, until 2004, when speeds were capped to limit heat. As computers increase in power and shrink in size, a new class of machines has emerged roughly every ten years (bottom).





©nature

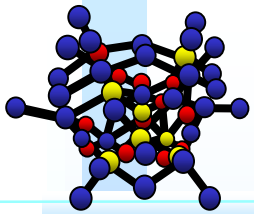


Frieda's Application …

Simulate the behavior of **F**(*x*,*y*,*z*) for 20 values of *x*, 10 values of *y* and 3 values of *z* (20*10*3 = 600 combinations)

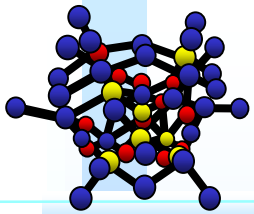
- **F** takes on the average 6 hours to compute on a "typical" workstation (total = 3600 hours)
- **F** requires a "moderate" (128MB) amount of memory
- **F** performs "moderate" I/O - (*x*,*y*,*z*) is 5 MB and **F**(*x*,*y*,*z*) is 50 MB

Non posso aspettare 3600 ore



Non posso aspettare 3600 ore

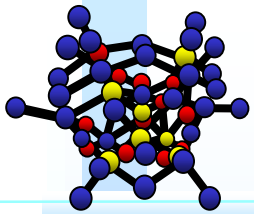
- Potrei eseguire le 600 **F**(*x*,*y*,*z*) su 600 calcolatori
- Costo di comunicazione dei dati
- Possibile se le 600 **F**(*x*,*y*,*z*) sono esecuzioni completamente indipendenti (come in questo caso),
  - dove ogni processo parte da dati che non dipendono dai risultati degli altri processi, delle altre esecuzioni



Architetture Parallele

- Obiettivi:
  - Ridurre i tempi di esecuzione
- Altri obiettivi
  - Trovare il miglior compromesso fra i costi della soluzione ed il tempo di esecuzione

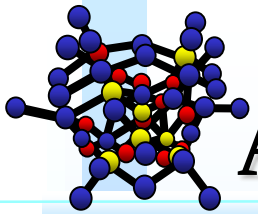
- ***Trasformare gli algoritmi seriali (approccio matematico) ad algoritmi parallele e/o distribuiti***
- ***Uso di compilatori ottimizzati***



Architetture Parallele

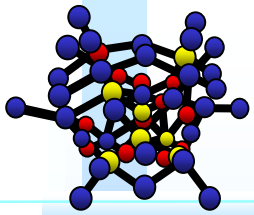
- La definizione di un'architettura ottima in termini di processi paralleli per il calcolo scientifico dipende dal problema
- Non confondiamo il Problema con la Soluzione

- Vi sono **problemi**
  - intrinsecamente sequenziali
  - Lineari vettoriali
  - Multidimensionali vettoriali
  - Paralleli nei dati di ingresso
  - Paralleli nei dai di uscita
  - Paralleli nei servizi
  - Paralleli nella procedura
  - Etc..



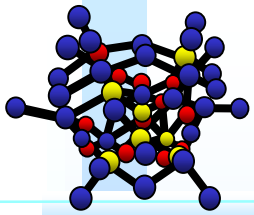
Architetture Parallele (Class. Flynn)

- **SISD**: Single instruction stream, single data stream
  - E.g.: un computer monoprocessore, monocore
- **SIMD**: Single instruction stream, multiple data stream
  - E.g.: le stesse istruzioni su tutti I nodi computazionali allo stesso tempo, ma lavorando su dati diversi, per esempio GPU su immagini
- **MISD**: Multiple instruction stream, single data stream
  - E.g.: ogni nodo puo' eseguire processi indipendenti ma sullo stesso stream di dati, un risultato singolo (poco realistico)
- **MIMD**: Multiple instruction stream, multiple data stream
  - E.g.: ogni nodo puo' eseguire processi indipendenti su dati diversi
  - Tipicamente la soluzione piu' utilizzata per il sistemi general purpose, cloud, etc.



Classificazione

- **SISD**: Single instruction stream, single data stream
  - Von Neumann
- **SIMD**: Single instruction stream, multiple data stream
  - Vector processor, Array processor
- **MISD**: Multiple instruction stream, single data stream
  - poco realistico

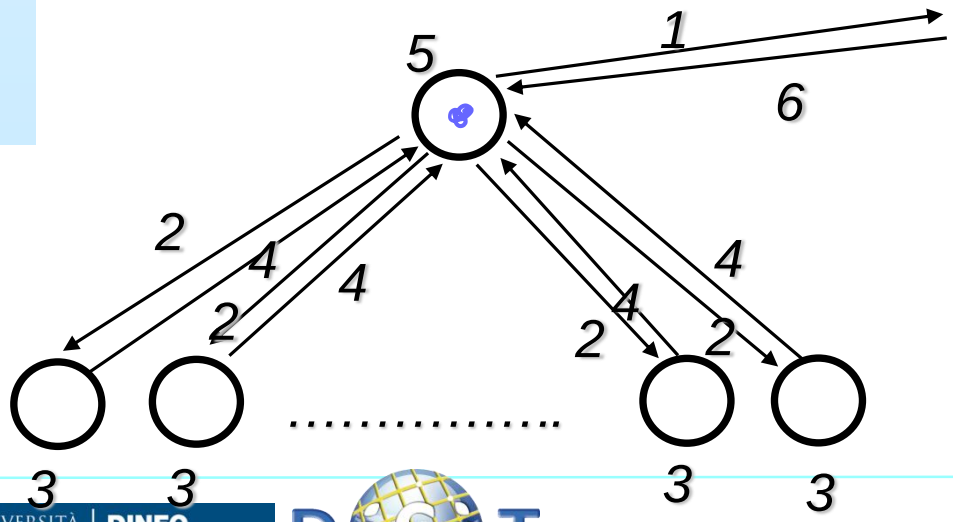


MIMD: Multiple instruction stream, multiple data stream

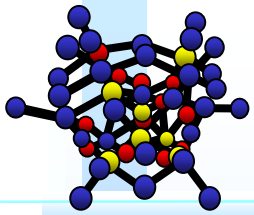
- **MultiProcessore**
  - Parallelismo interno al calcolatore
  - Memoria condivisa, variabili condivise
  - Sincronizzazioni
  - E.g., Uniform Memory Access: XEON
- **MultiComputer**
  - Cloud, architetture parallele
  - Comunicazione tramite canali dedicati,
  - Message passing, Send e Receive
    - Hypercubes
- **Sistemi Distribuiti**,
  - Cluster, Massive parallel processor
    - ➔GRID, cloud, ..

Esempio di caso Lineare

- VettC = VettA + VettB
- In modo sequenziale il Costo e' o(N), in parallelo il costo e' 1
- Soluzione parallela:
  - N nodi
  - Un concentratore per raccolta dati
  - Comunicazione fra nodi: assente
  - Comunicazione con il nodo concentratore

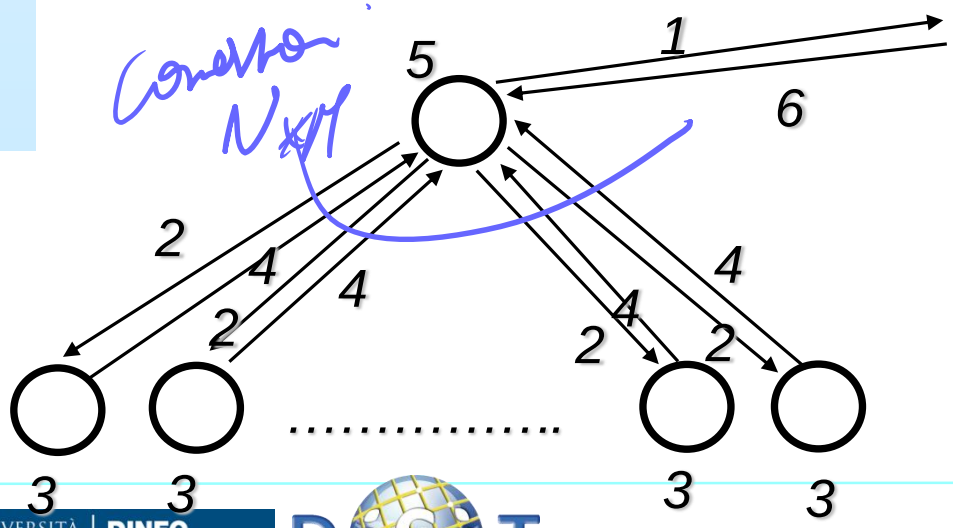


1. *Passa A e B*
2. *Passa Ai, Bi*
3. *Calcola Ai+Bi*
4. *Passa Ci*
5. *Metti insieme C*
6. *Passa C*

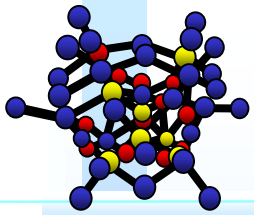


Esempio di caso 2D, (..nD)

- MatC = MatA + MatB
- In modo sequenziale il Costo e' o(NM), in parallelo il costo e' 1
- Soluzione parallela:
  - N*M nodi
  - Un concentratore per raccolta dati
  - Comunicazione fra nodi: assente
  - Comunicazione con il nodo concentratore



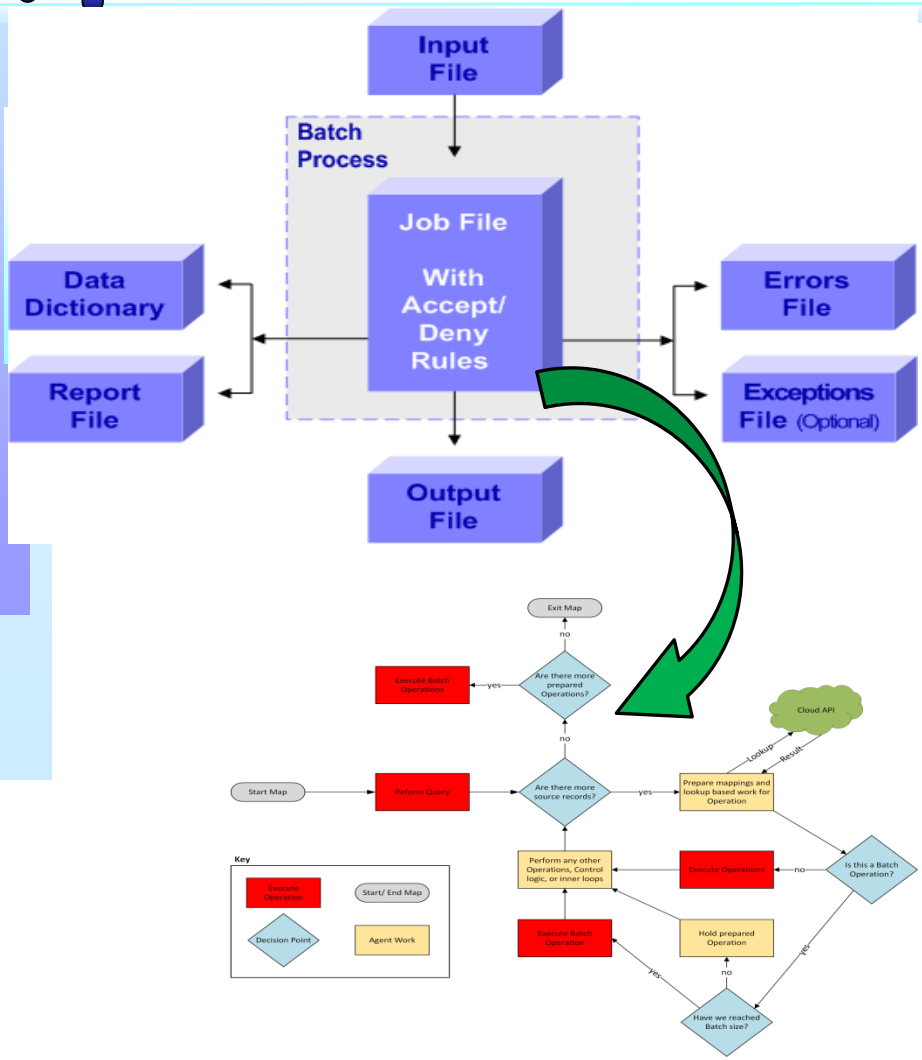
a. *Passa A e B*
b. *Passa Aij, Bij*
c. *Calcola Aij+Bij = Cij*
d. *Passa Cij*
e. *Metti insieme C*
f. *Passa C*



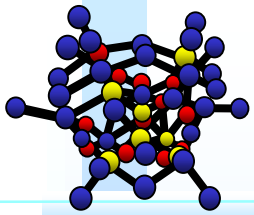
Comunicazione fra processi

- In alcuni casi vi è la necessità di effettuare connessioni/comunicazioni dirette fra modi dell'architettura parallela
- Se queste comunicazioni possono essere eseguite in parallelo (contemporaneamente) si risparmia tempo rispetto a farle gestire tutte da un nodo centrale come in molti sistema di gestione di processi concorrenti
- Un sistema di **gestione di processi concorrenti** dovrebbe
  - permettere di mappare in modo logico un'architettura parallela/concorrente qualsiasi sull'architettura fisica in termini di processori e calcolatori
  - Permettere ai nodi (processori) di comunicare chiamandosi in modo logico e non fisico
  - Permettere di identificazione in modo logico i nodi.

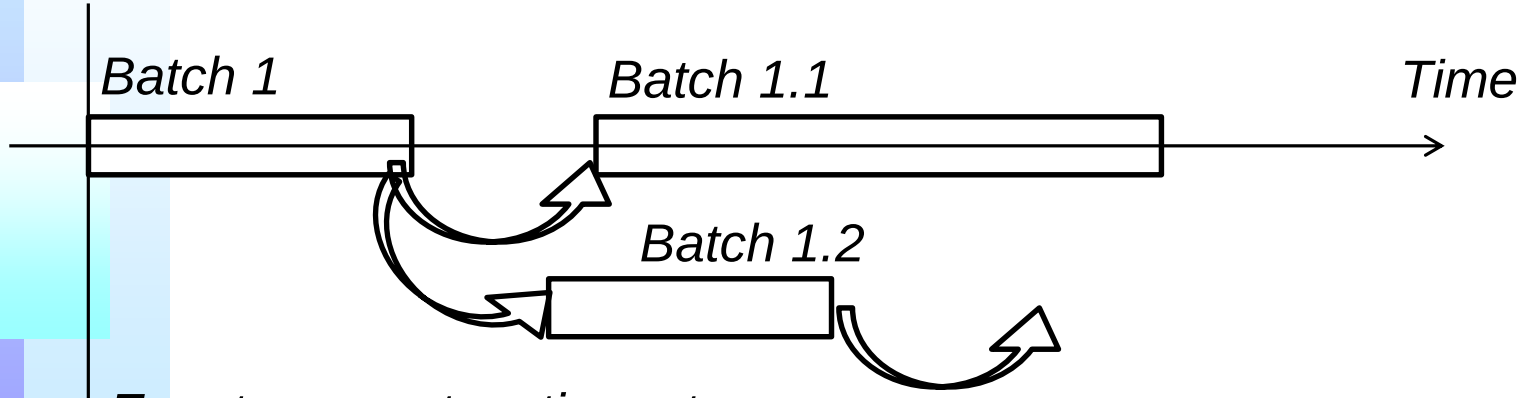
Batch Processing



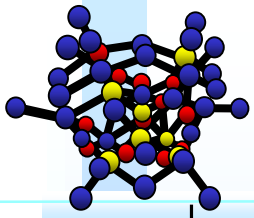
- Script language
- Smilar to workflow or flowchart
- Sequence of Commands
- Intermediate status on disk or memory
- Executed command driven:
  - On demand, sporadically
  - Periodically



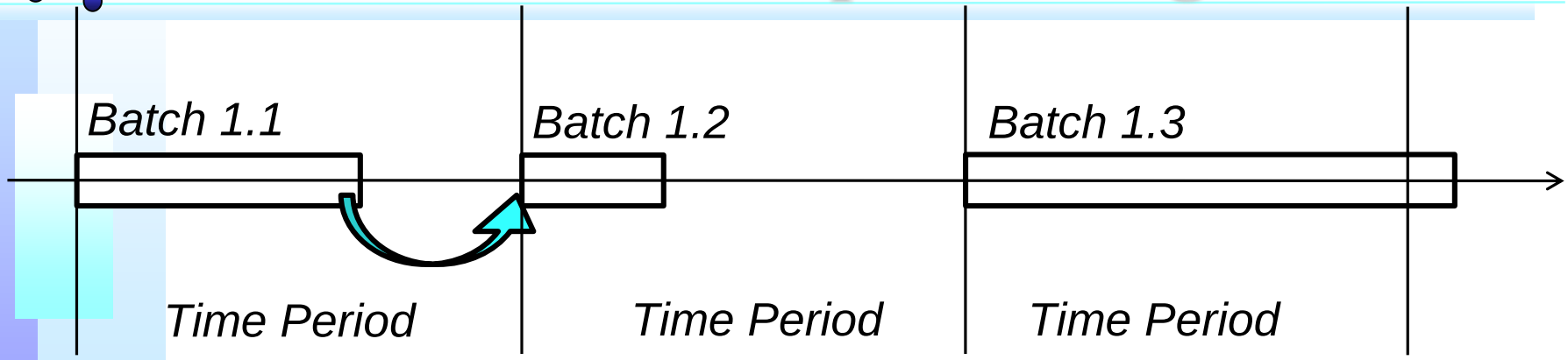
aPeriodic Batch processing



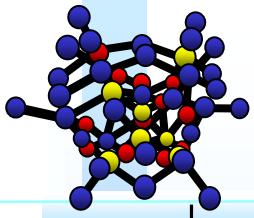
- Activated/Fired/Triggered by events, on demand, etc..
  - Synchronous with something
  - chained or not
- May fire/generate (ask to do or directly do) other jobs/batches:
  - on the same or different computers
  - Identical or different
- Activation may be asked to a third party manager
- Duration of execution depending on data !



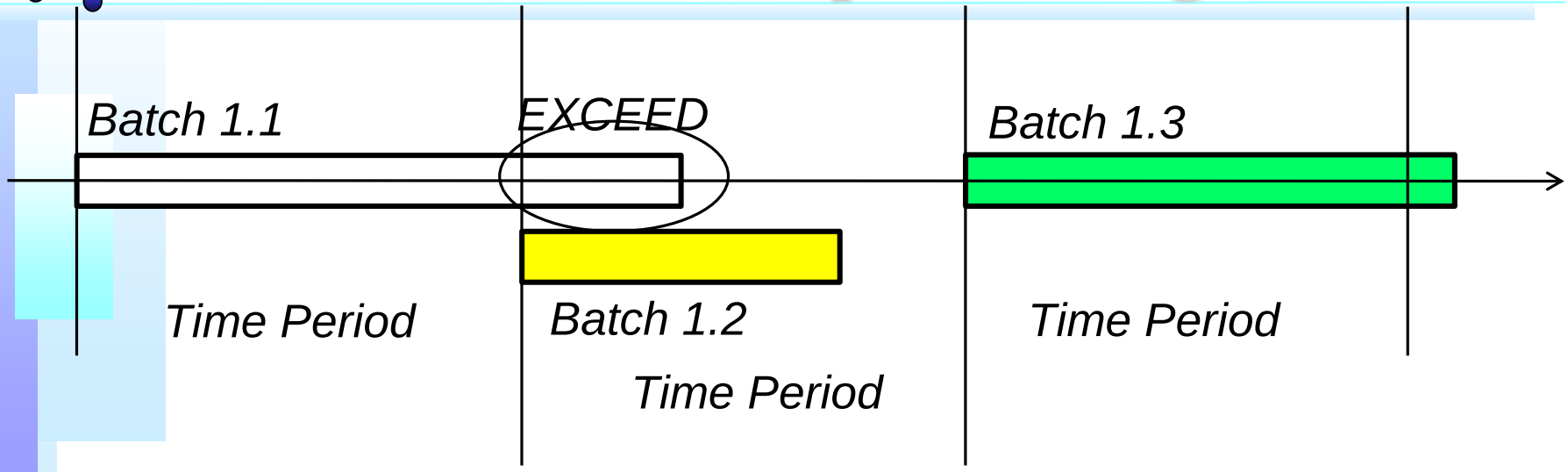
Periodic Batch processing



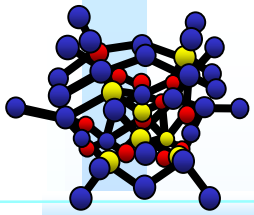
- Scheduled on periodic Firing time event (with a validity period of firing conditions from xx to yy)
  - synchronous
- Execution time duration depending on data or other..
  - Execution may exceed the Time Period
- Each single execution MAY or MAY NOT depend on the preceding one



Periodic Batch processing

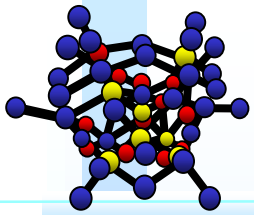


- If the Execution Time Duration exceeds the Time Period
  - A) the successive execution (Batch 1.2) is overlapped (yellow)
    - If it happens systematically: the number of tasks grow indefinitely consuming all resources → until crash
  - B) the successive execution (Batch 1.2) is canceled to wait for the next one (Batch 1.3, green),
    - skipping the second execution, Batch 1.2



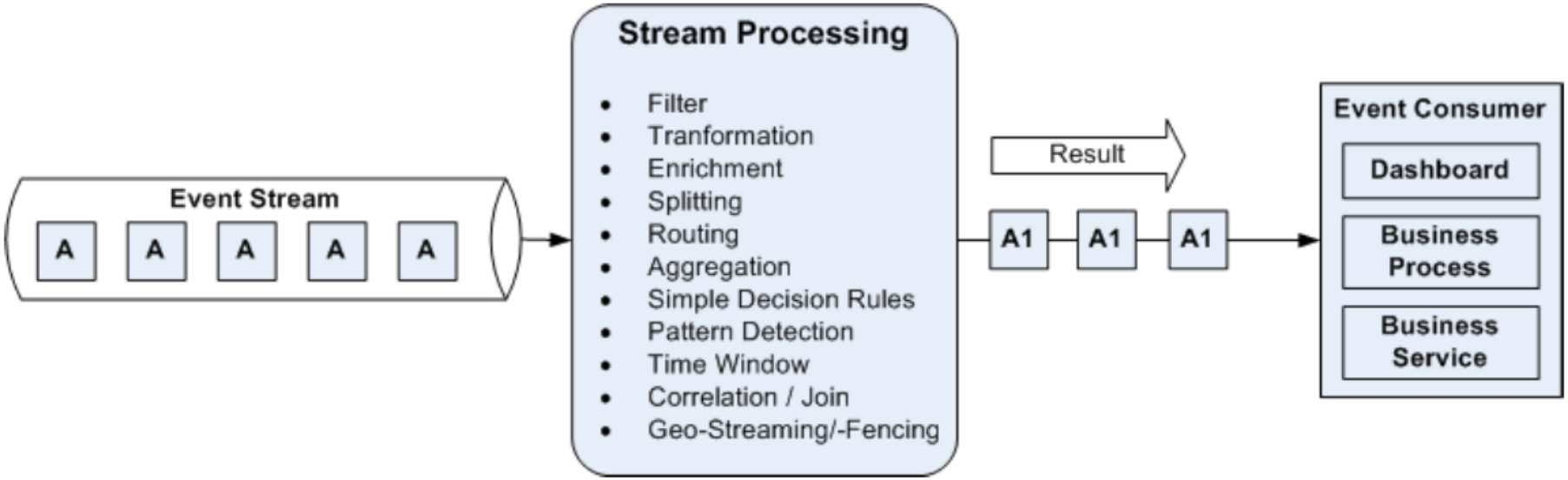
Stream Processing

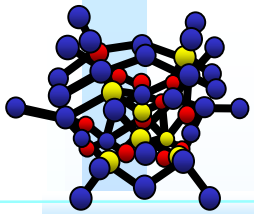
- paradigma di programmazione parallela
  - Detto anche di real time processing
- Lavora con:
  - Dati parziali e non su tutto l'insieme dei dati.
  - Per esempio: valutare il contenuto spettrale di un segnale:
    - Su tutto il segnale
    - Su una finestra temporale di 30 secondi, per ogni secondo un nuovo valore, small delay/latency, (but present , see pipeline)
- L'attivazione del processo corrisponde spesso all'arrivo del dato nello stream, nella pipeline,
  - si ha sincronizzazione e comunicazione in un solo colpo.
  - Tipicamente una sequenza di azioni semplici attivate da
    - l'arrivo delle condizioni e dei dati per calcolare i risultati
    - ed in modo asincrono
    - tipicamente senza avere necessità di memorizzare il dato
  - → elimina alcuni problemi della programmazione parallela.



Stream Processing

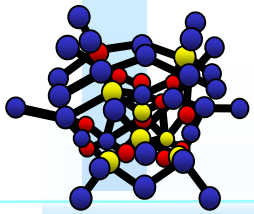
- Example on languages
  - SISAL (Streams and Iteration in a Single Assignment Language)
  - CUDA (Compute Unified Device Architecture) for NVIDIA GPU
- Applications:
  - IOT, media recognition, log processing, social media enrichment, indexing,





sommario

- Contesto tecnologico
- Overhead e Speed-UP
- Aspetti Topologici per il calcolo parallelo
- Soluzioni GRID
- Soluzioni MicroGRID
- Apache Hadoop MapReduce
  - Vedere anche esercitazioni
  - http://www.disit.org/7073
- Apache Hadoop MapReduce 2
- Apache Spark



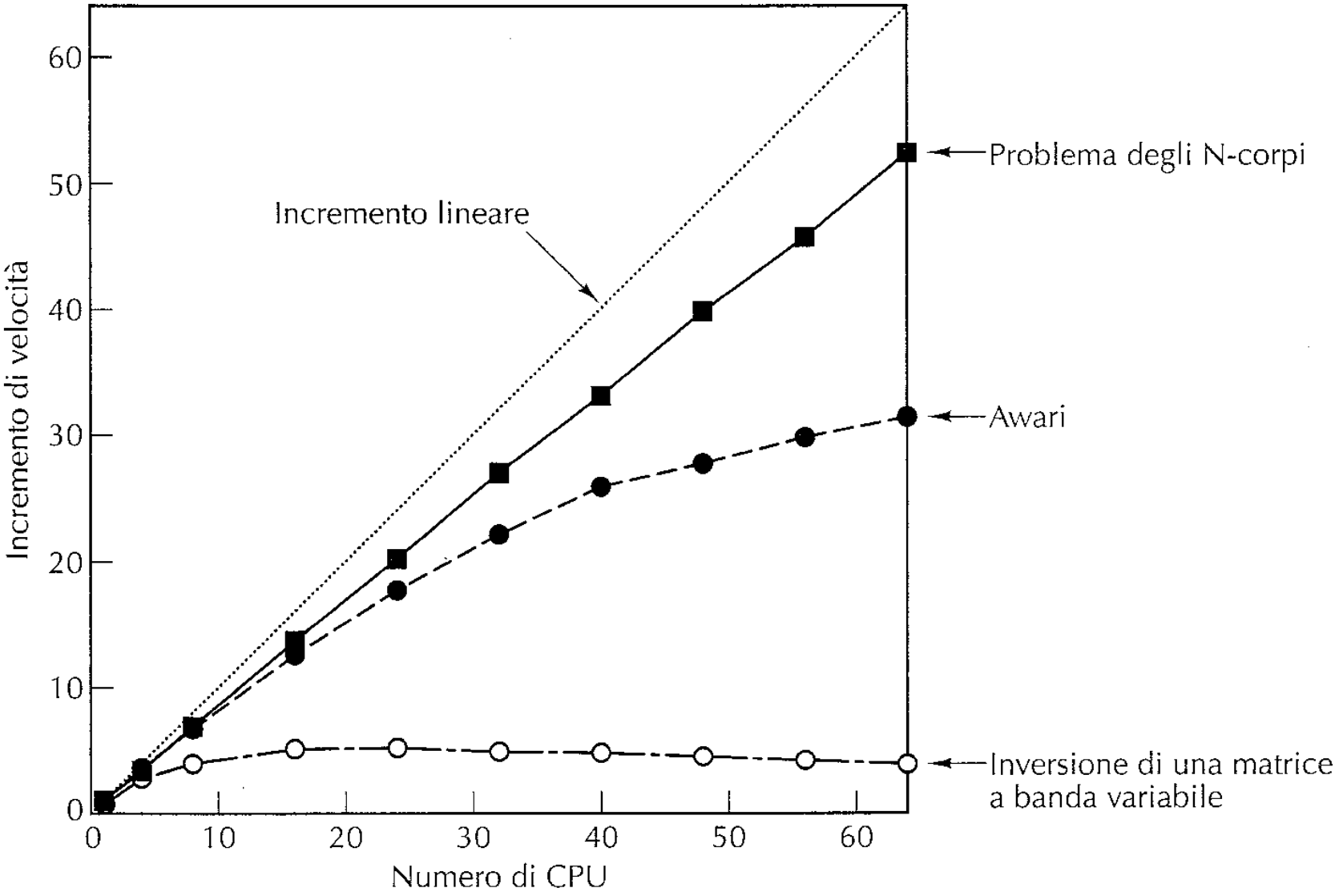
overhead

- Nel caso lineare Costo computazionale
  - O(n)
- Nel caso parallelo
  - Comunico Ai e Bi: O(n)+O(n)
  - Calcolo Ci: O(1)
  - Comunico Ci: O(n)

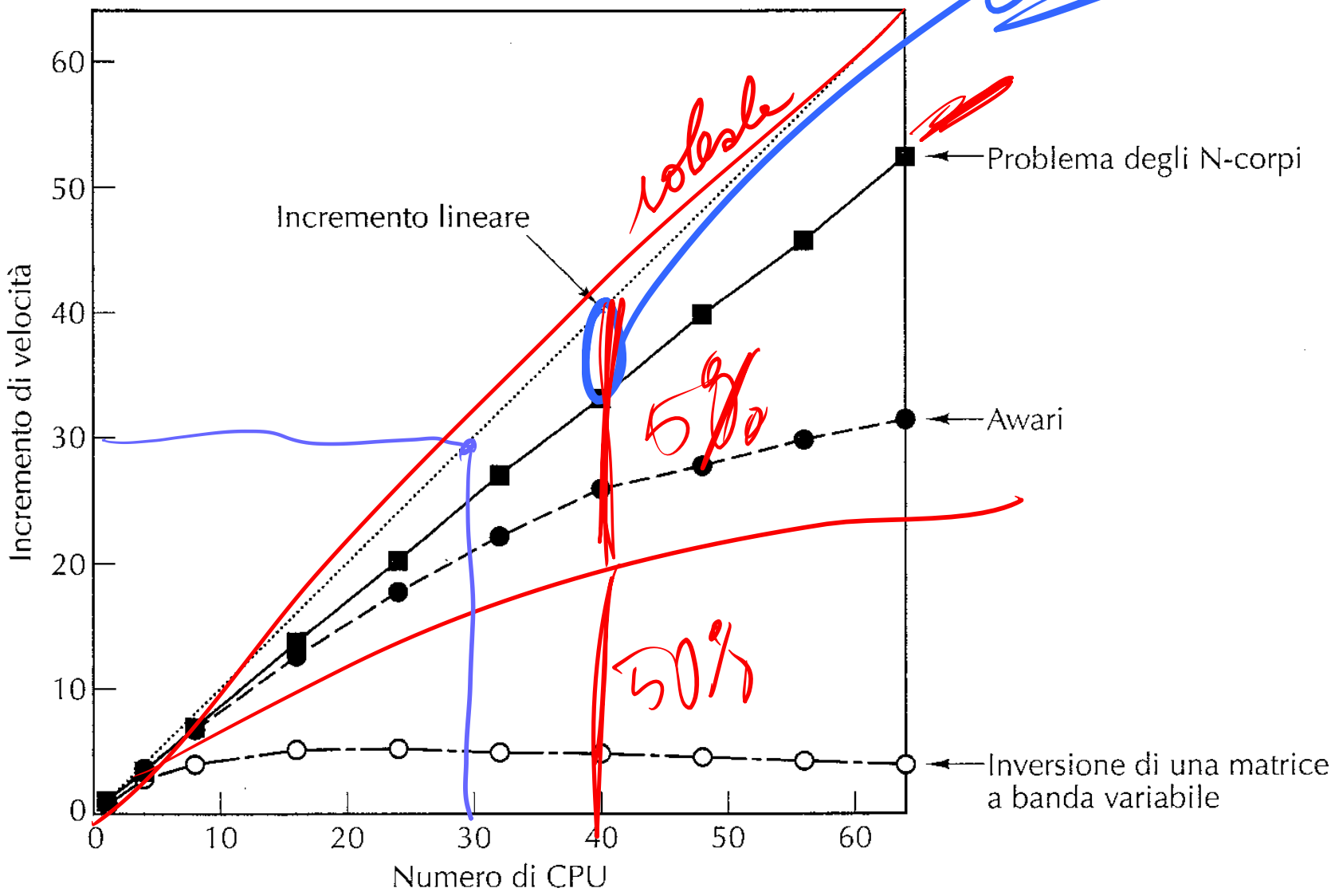
- Quale delle due soluzioni e' piu' efficiente?
  - Dipende dai dati, da N, dal costo della comunicazione, etc.

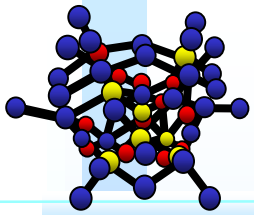
- Architetture specifiche (composizioni di processori con connessioni dedicate) possono produrre soluzioni efficaci per problemi specifici.

Speed Up

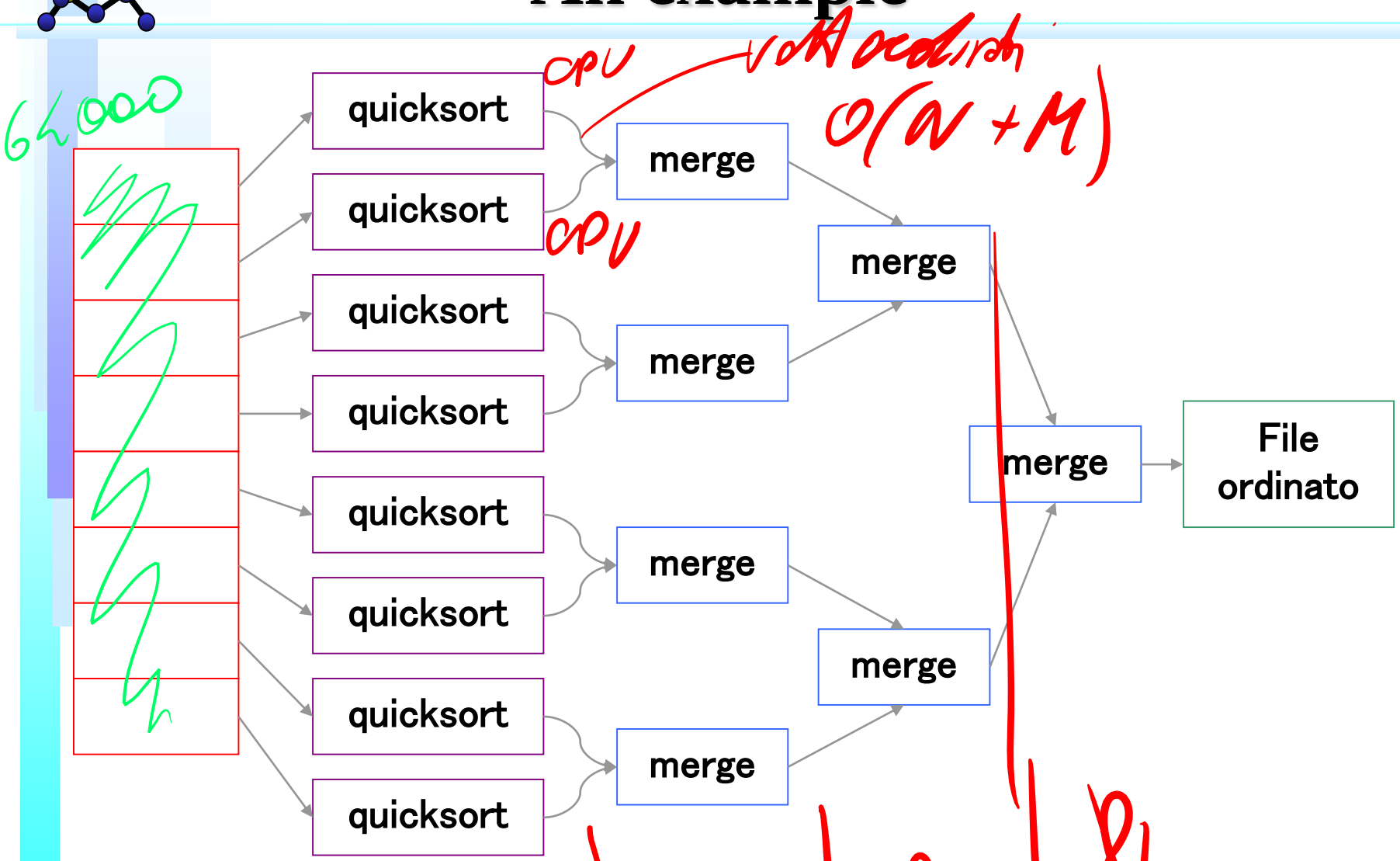

Incremento di velocità
Numero di CPU
0
10
20
30
40
50
60
Incremento lineare
Problema degli N-corpi
Awari
Inversione di una matrice
a banda variabile

Speed Up

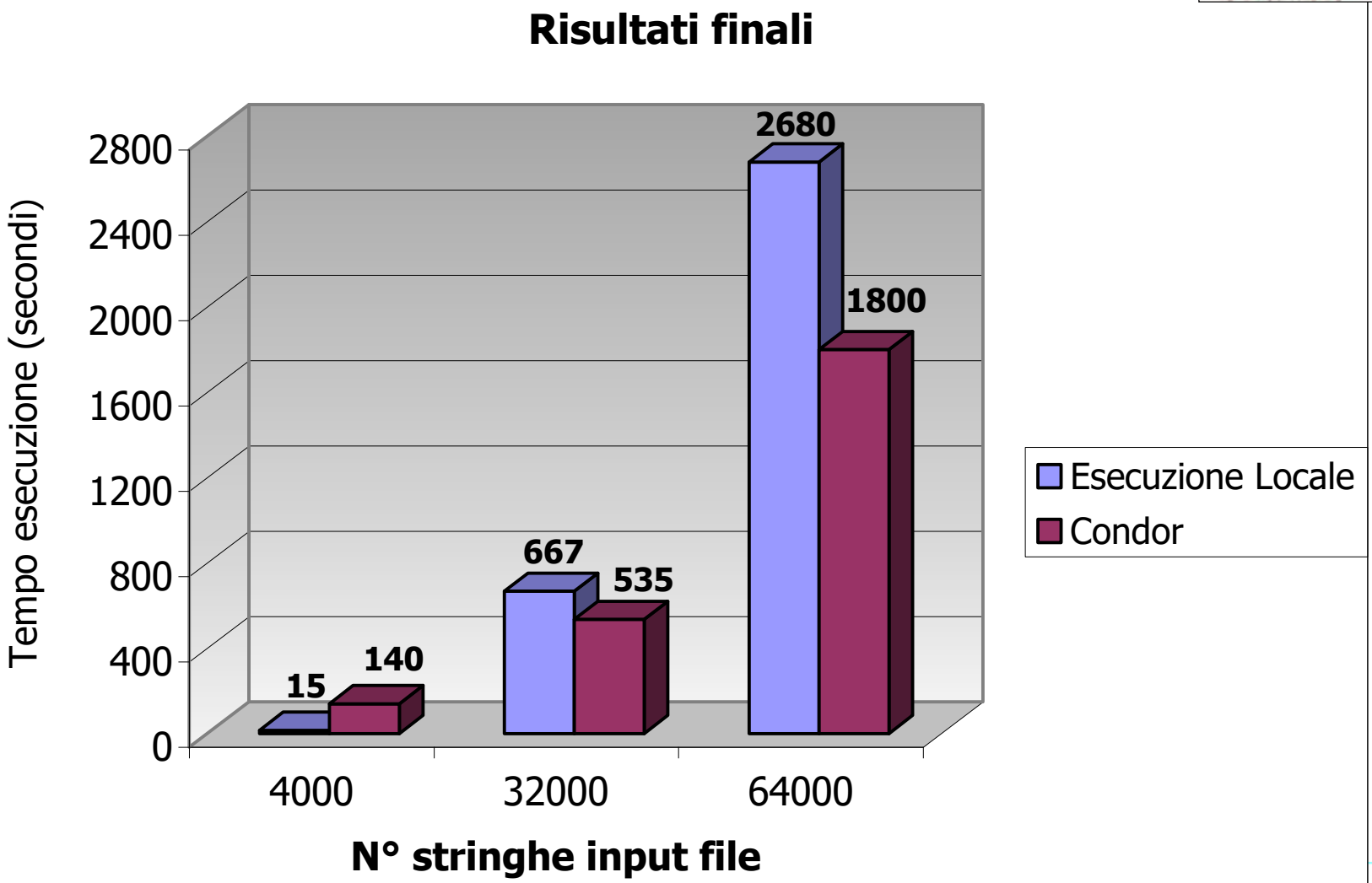

Incremento di velocità
Numero di CPU
0
10
20
30
40
50
60
Incremento lineare
Problema degli N-corpi
Awari
Inversione di una matrice
a banda variabile
overhead
50%

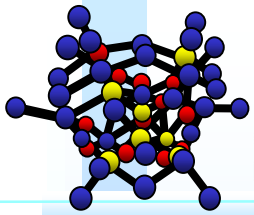


An example


quicksort
quicksort
quicksort
quicksort
quicksort
quicksort
quicksort
quicksort
merge
merge
merge
merge
merge
merge
File ordinato

Un Esperimento CONDOR at DISIT



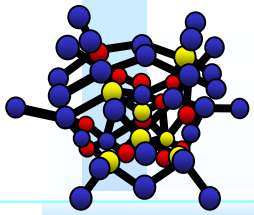


Scelte che hanno condizionato il risultato

- Non si è utilizzato un *merge sort* dall'inizio perché' non è efficiente visto che inviare due valori ad un nodo per sapere quale dei due è maggiore costa di più che farlo in locale
  - Andamento del costo locale e distribuito del merge, per decidere

- **Si poteva utilizzare:**
  - Algoritmi di ordinamento diversi
  - Una partizione diversa dei dati, non 8 processi ma per esempio 4, con due livelli e non 3

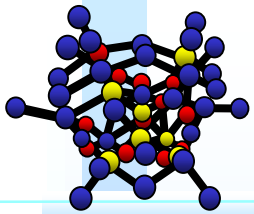
In alcuni casi e configurazioni si sarebbe potuto ottenere un Speed Up più elevato, un minor over head percentuale



Allocazione dei processi

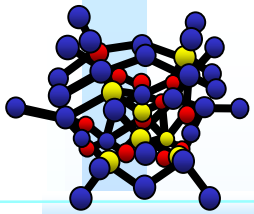
- **Caso 1, caso semplice ed immediato:**
  - Ho 8+4+2+1 processori e un processo per processore.
  - Ogni processo sa da dove prendere i dati e dove mandare i risultati
  - Costo di comunicazione elevato

- **Caso 2, riduzione del costo di comunicazione:**
  - Ho 8 processori, questi computano la prima fase, poi quelli pari comunicano il risultato ai dispari, che
  - divengono pertanto i 4 processori della fase 2
  - Quelli che hanno id divisibile per 4 passano i risultati a quelli modulo 4 + 1, questi divengono quelli della fase 3
  - Etc. etc.



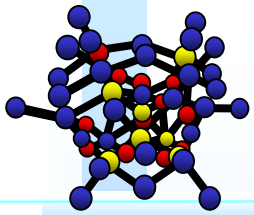
sommario

- Contesto tecnologico
- Overhead e Speed-UP
- Aspetti Topologici per il calcolo parallelo
- Soluzioni GRID
- Soluzioni MicroGRID
- Hadoop MapReduce
  - Vedere anche esercitazioni
- Apache Spark



Sistemi Multicomputer

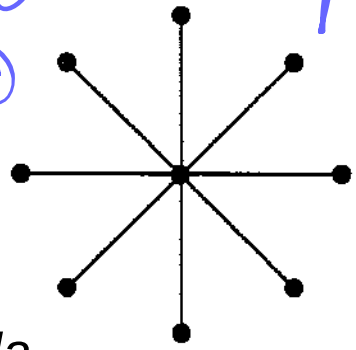
- Tecnologie di Interconnesione
- Ogni nodo è connesso con gli nodi in base ad un certa topologia, tramite canali dedicati ad alte prestazioni
- Topologie come:
  - Stella, anello, mesh, toro, cubo, ipercubo
- Si implementano soluzioni di sincronizzazione



Soluzioni parallele diverse

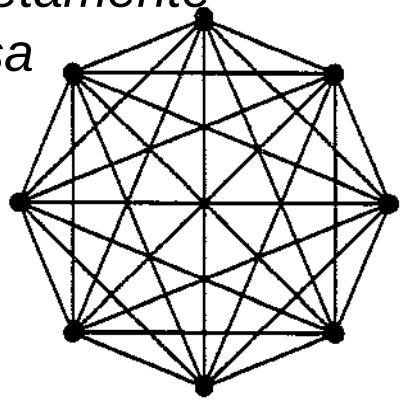
topologie

- stella

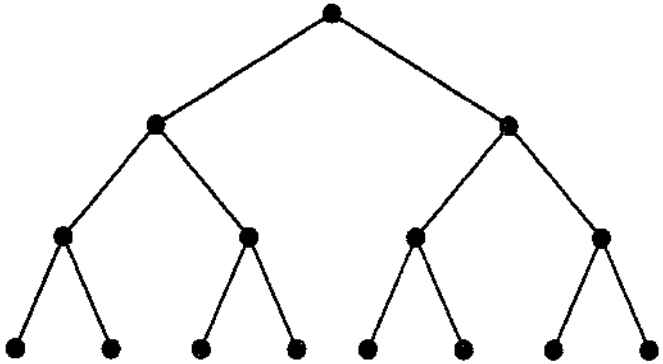


(a)

- Completamente connessa

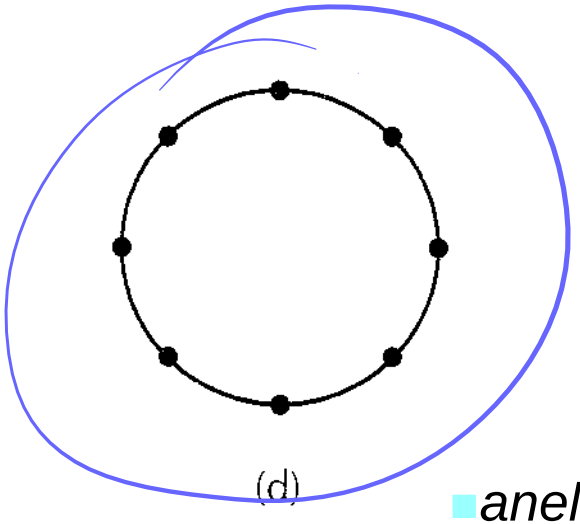


(b)



(c)

- gerarchica

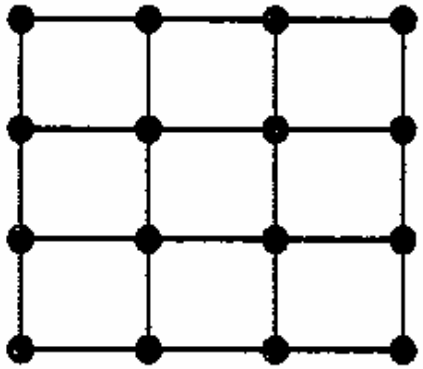


(d)

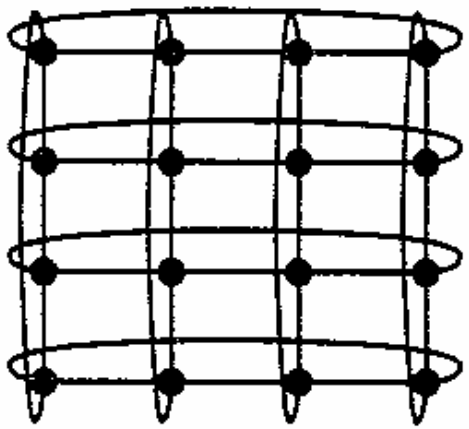
- anello

Soluzioni parallele diverse topologie

- *mesh*



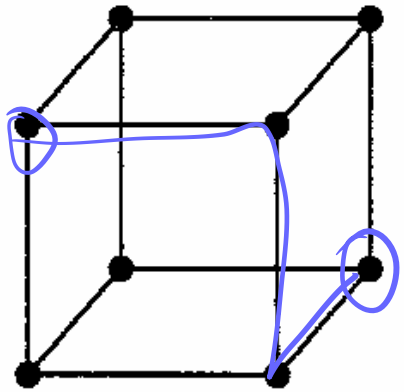
(e)



(f)

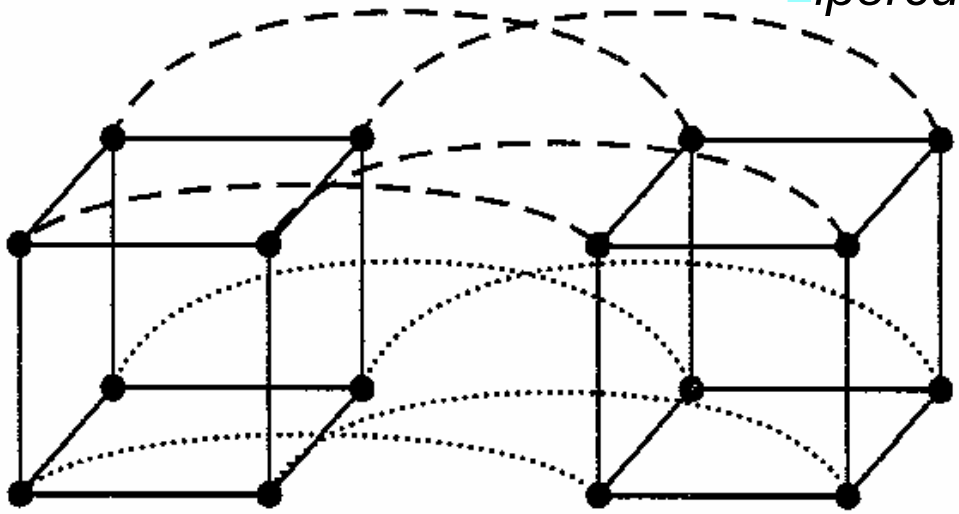
- *Mesh*
- *richiusa*

- *cubo*



(g)

- *ipercubo*

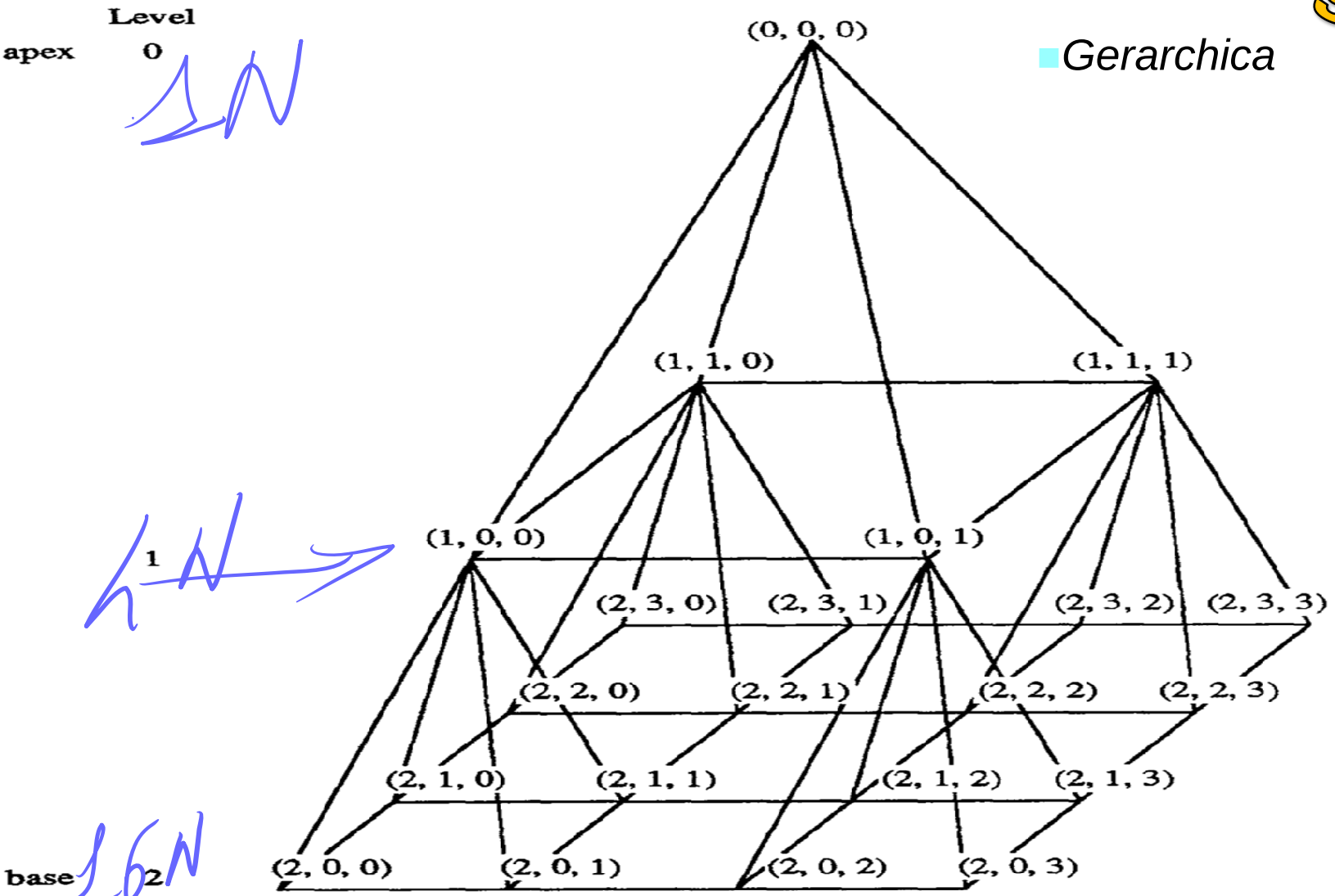


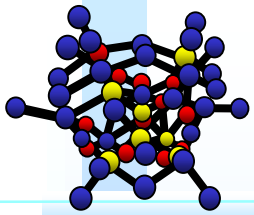
(h)

Piramide detta anche grid

topologie

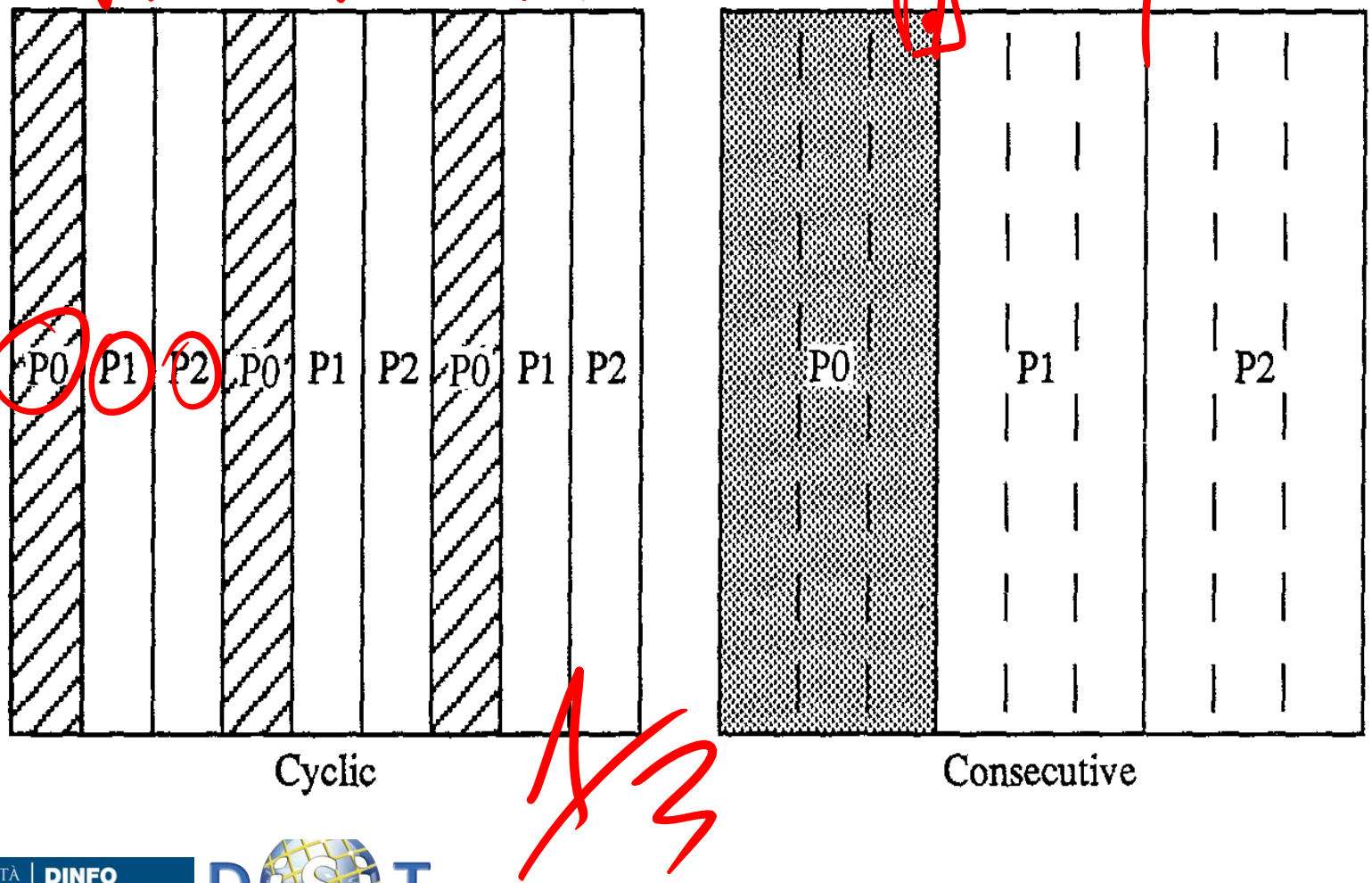
Gerarchica





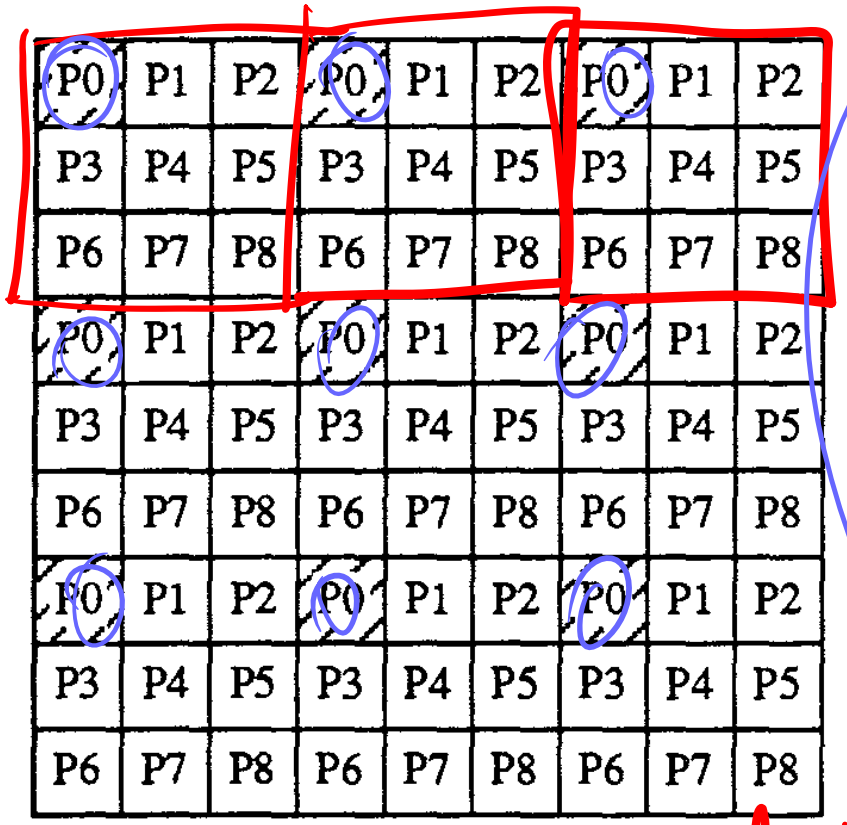
3 Processori in forma ciclica o consecutiva

Partizione dei dati

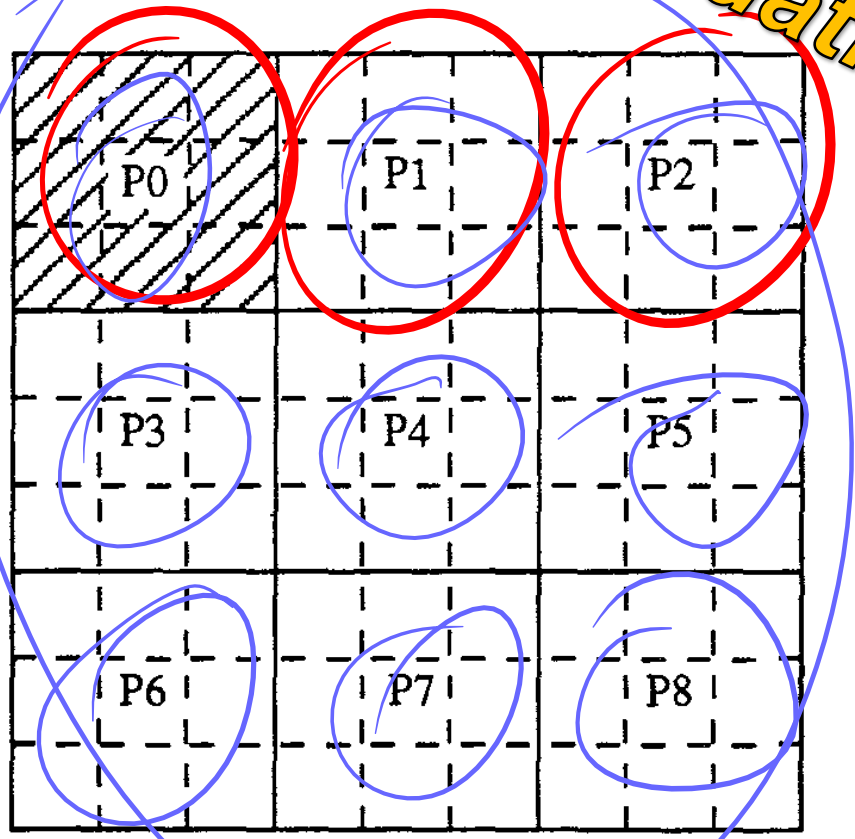


8 Processori in forma ciclica o consecutiva

Partizione dei dati



Cyclic



Consecutive

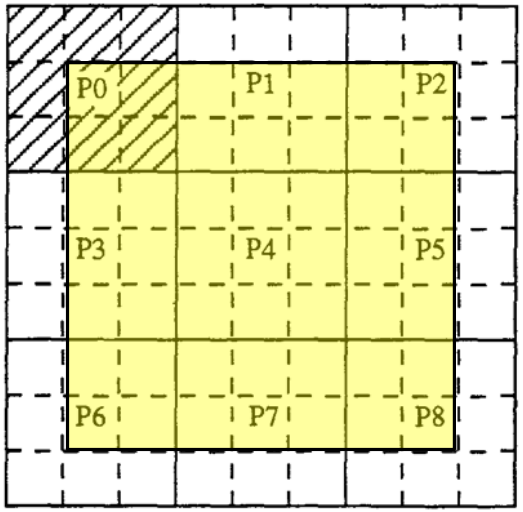
Esempio di elaborazione locale spaziale

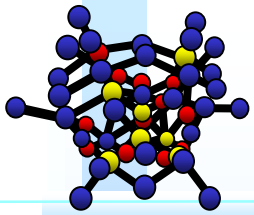
1	2	3
4	5	6
7	8	9

1	2	3
4	5	6
7	8	9

1	2 1	3 2
4	5 4	6 5
7	8 7	9 8

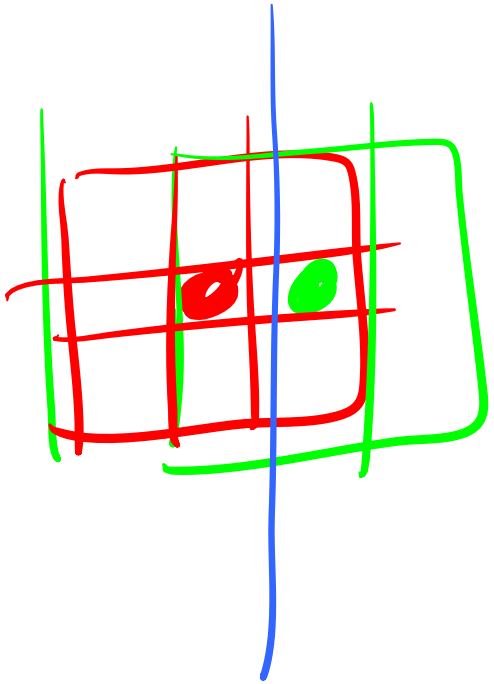
- *Media:* $= \sum_{i=1}^{9} Matrice\ (i)$
- *per stimarla per ogni punto della matrice:*
  - *Problemi al contorno*
  - *Sovrapposizione dei dati*

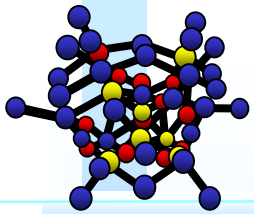




Comunicazioni fra processi

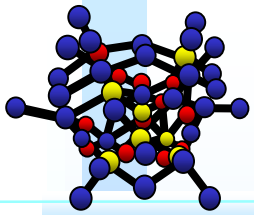
- **Comunicazione fra processi** per congiungere dati parziali
  - Spesso necessarie per processi iterativi
  - Soluzioni di equazioni alle derivate parziali
  - Soluzioni agli elementi finiti
  - Inversioni di matrici a blocchi
- **Condizioni al contorno**
  - Soluzioni di equazioni alle derivate parziali
  - Soluzioni agli elementi finiti
- **Integrazione del calcolo**
  - Equazioni differenziali alle derivate parziali
  - Calcolo di Flussi
  - Calcolo per antenne





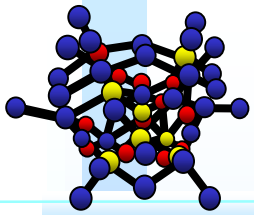
Problemi

- **Parallelizzazione degli algoritmi**
  - Progettazione parallela
  - Non tutti gli algoritmi si possono parallelizzare in modo facile e poco costoso..
  - Bilanciamento fra vantaggi e costi di comunicazione
  - Massimizzazione dello Speed Up:
    - Efficienza della soluzione parallela
- **Allocazione ottima dei processi sui nodi/peer:**
  - Capacità dei nodi può cambiare nel tempo (Clock, rete, memoria, storage)
  - Costi di comunicazione che cambiano nel tempo, altre comunicazioni
  - Problema di allocazione:
    - Genetic Algorithms, Taboo Search, etc.
- **Tolleranza ai fallimenti**
  - Ridondanza dei processi
  - Migrazione dei processi, salvataggio del contesto
- **Limitato controllo delle capacità dei nodi**
- **Limitato controllo delle prestazioni: CPU, rete, QoS, ….**

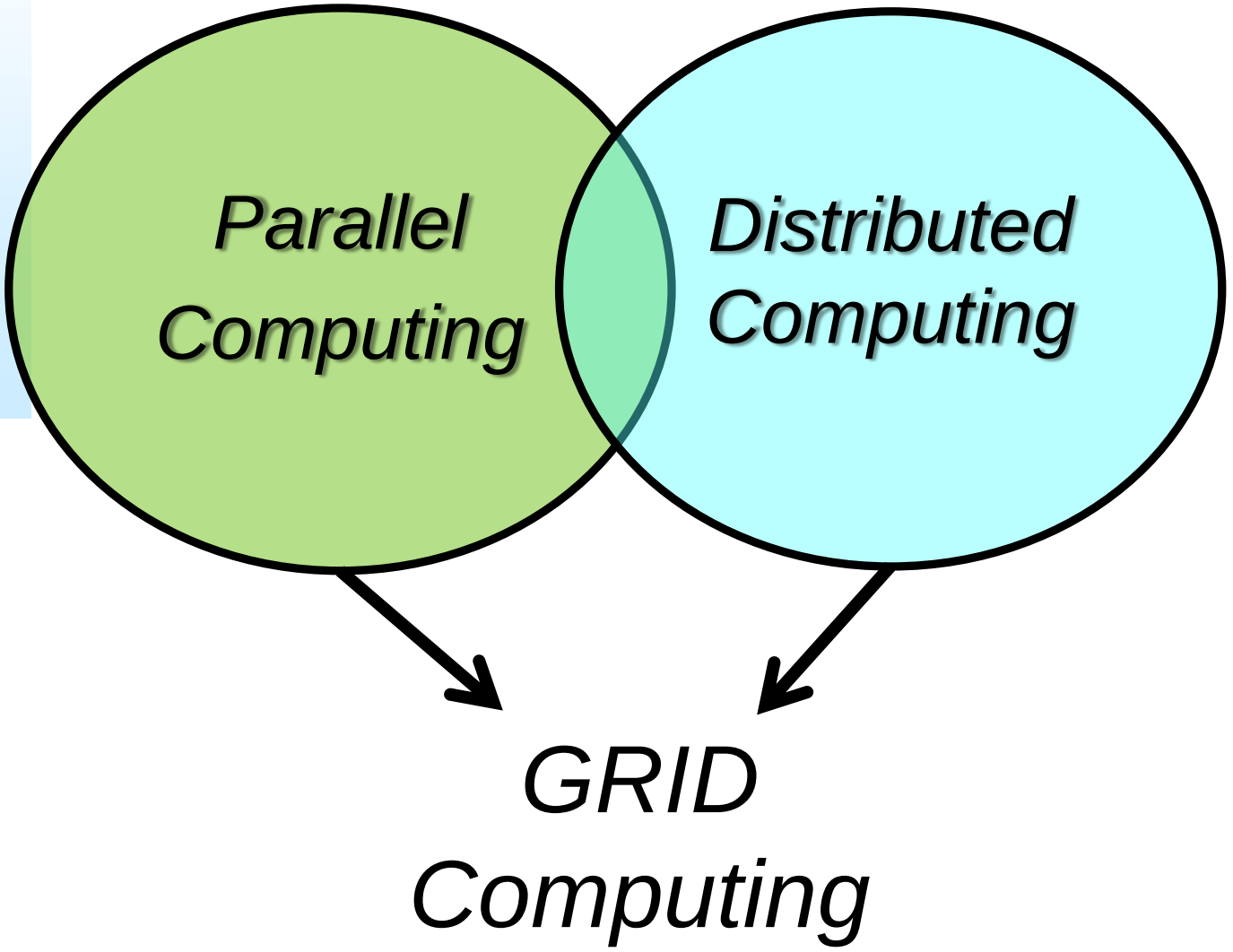


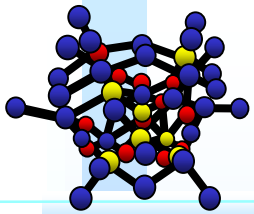
sommario

- Contesto tecnologico
- Overhead e Speed-UP
- Aspetti Topologici per il calcolo parallelo
- Soluzioni GRID
- Soluzioni MicroGRID
- Apache Hadoop MapReduce
  - Vedere anche esercitazioni
  - http://www.disit.org/7073
- Apache Hadoop MapReduce 2
- Apache Spark



Grid vs Distributed and Parallel

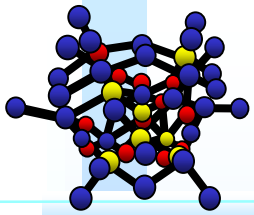


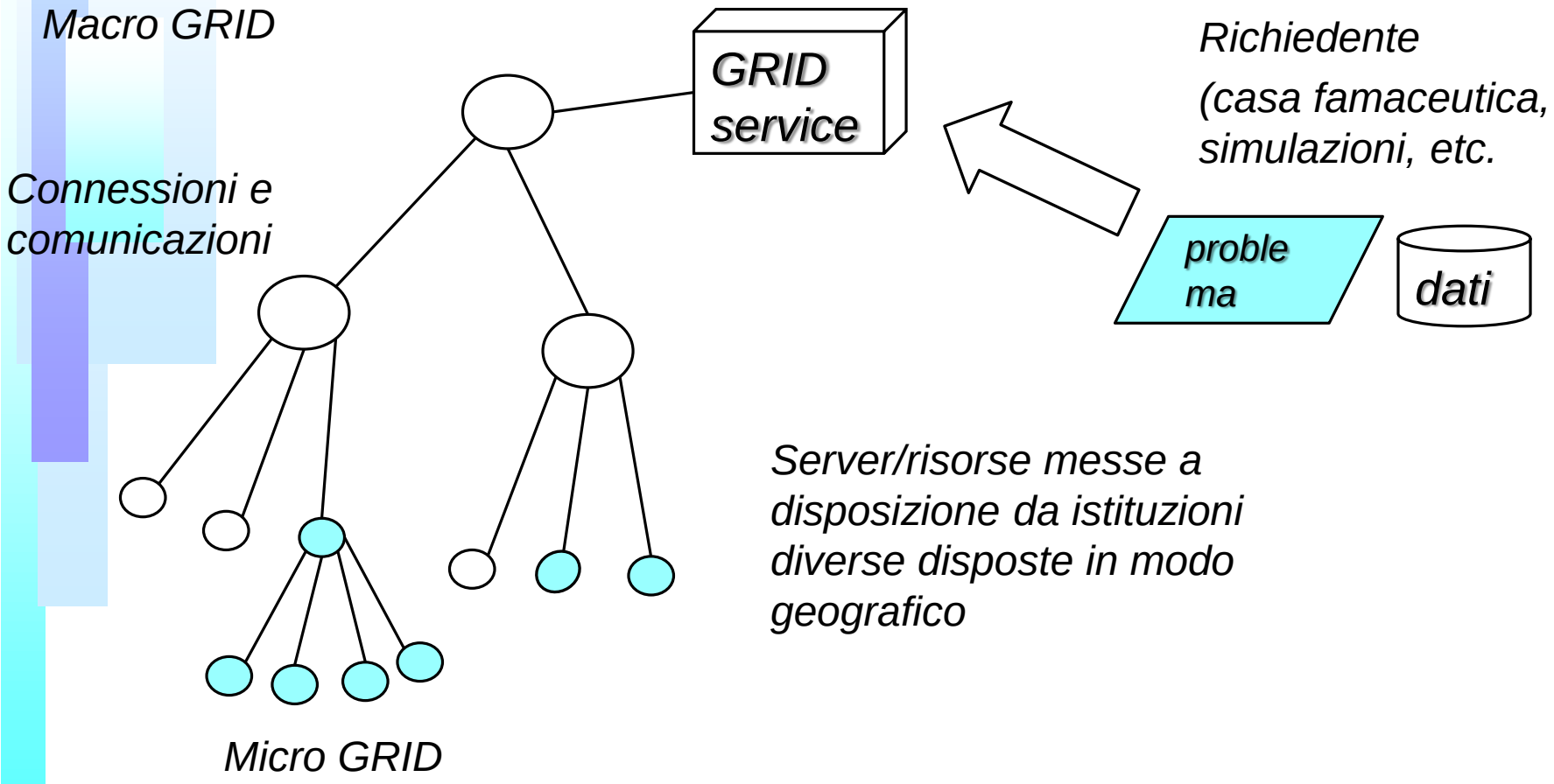


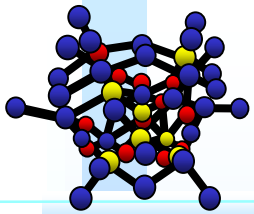
The GRID

- "the **Grid**" term coined in the mid 1990s to denote a distributed computing infrastructure for advanced science and engineering
- "Resource sharing & coordinated problem solving in dynamic, multi-institutional virtual organizations" (Ian Foster, Karl Kesselman)

- Un insieme di risorse computazionali, di dati e reti appartenenti a diversi domini amministrativi
- Fornisce informazioni circa lo stato delle sue componenti tramite Information Services attivi e distribuiti.
- Permette agli utenti certificati di accedere alle risorse tramite un'unica procedura di autenticazione
- Gestisce gli accessi concorrenti alle risorse (compresi i fault)
  - No single point of failure

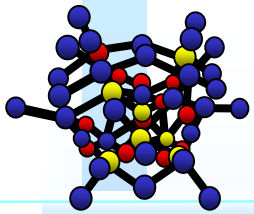



Macro GRID
GRID service
Richiedente
(casa famaceutica, simulazioni, etc.
Connessioni e comunicazioni
proble ma
dati
Server/risorse messe a disposizione da istituzioni diverse disposte in modo geografico
Micro GRID



Per essere un GRID

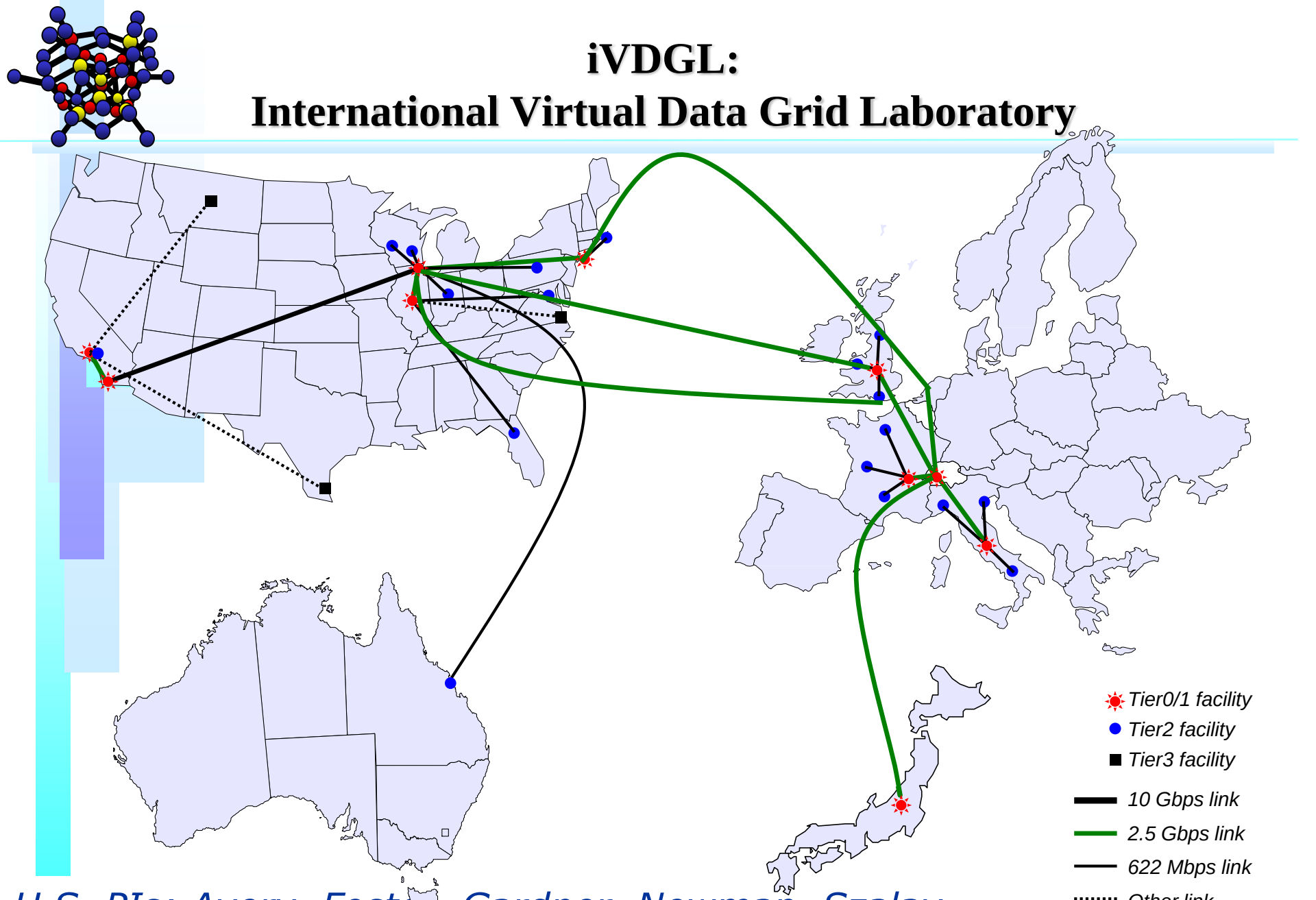
- **coordina risorse** e fornisce meccanismi di sicurezza, policy, membership…
- **Usa protocolli ed interfacce standard**, open e general-purpose.
- permette l'utilizzo delle sue risorse con diversi livelli di **Quality of Service** (tempo di risposta, throughput, availability, sicurezza…).



Concetti Estesi del GRID

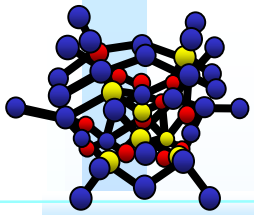
- **Virtual Organization** (VO) è costituita da:
  - un insieme di individui o istituzioni
  - un insieme di risorse da condividere
  - un insieme di regole per la condivisione
- VO: utenti che condividono necessità e requisiti simili per l'accesso a risorse di calcolo e a dati distribuiti e perseguono obiettivi comuni.
- abilità di negoziare le modalità di condivisione delle risorse tra i componenti una VO (providers and consumers) ed il successivo utilizzo per i propri scopi. [I.Foster]

iVDGL:
International Virtual Data Grid Laboratory



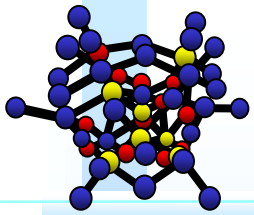
U.S. PIs: Avery, Foster, Gardner, Newman, Szalay

www.ivdgl.org



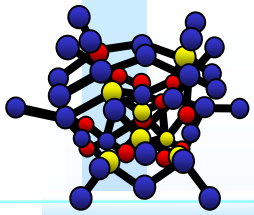
Grid of Cluster computing

- **GRID**
  - collezione di risorse distribuite, possibilmente eterogenee, ed una infrastruttura hardware e software per calcolo distribuito su scala geografica.
  - mette assieme un insieme distribuito ed eterogeneo di risorse da utilizzare come piattaforma per High Performance Computing.
- **Cluster, a micro-grid**
  - Usualmente utilizza piattaforme composte da nodi omogenei sotto uno stesso dominio amministrativo
  - spesso utilizzano interconnessioni veloci (Gigabit, Myrinet).
  - Le componenti possono essere condivise o dedicate.

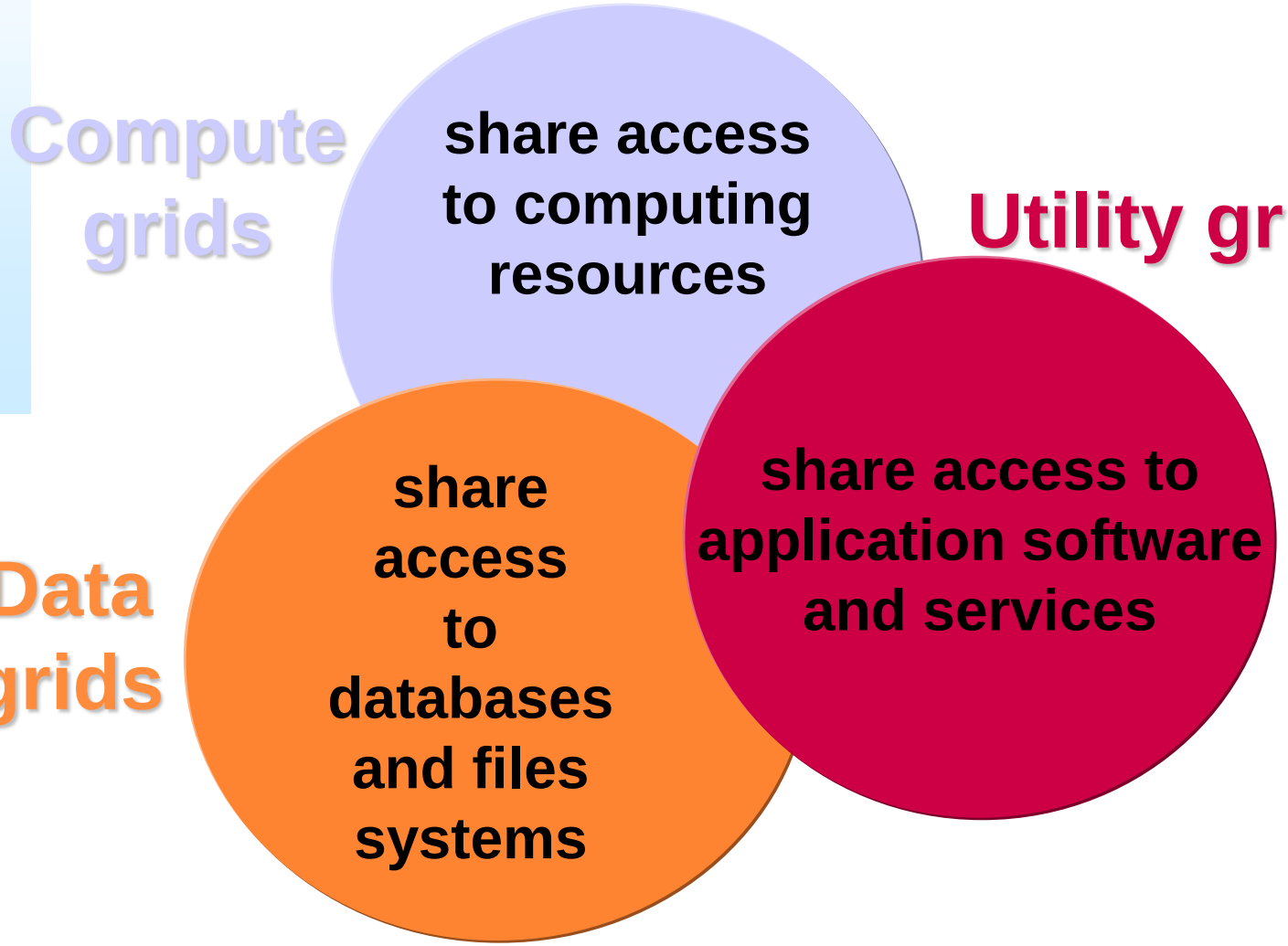


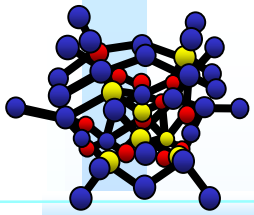
Scienze Data Intensive

- Fisica nucleare e delle alte energie
  - Nuovi esperimenti del CERN
- Ricerca onde gravitazionali
  - LIGO, GEO, VIRGO
- Analisi di serie temporali di dati 3D (simulazione, osservazione)
  - Earth Observation, Studio del clima
  - Geofisica, Previsione dei terremoti
  - Fluido, Aerodinamica
  - Diffusione inquinanti
- Astronomia: Digital sky surveys



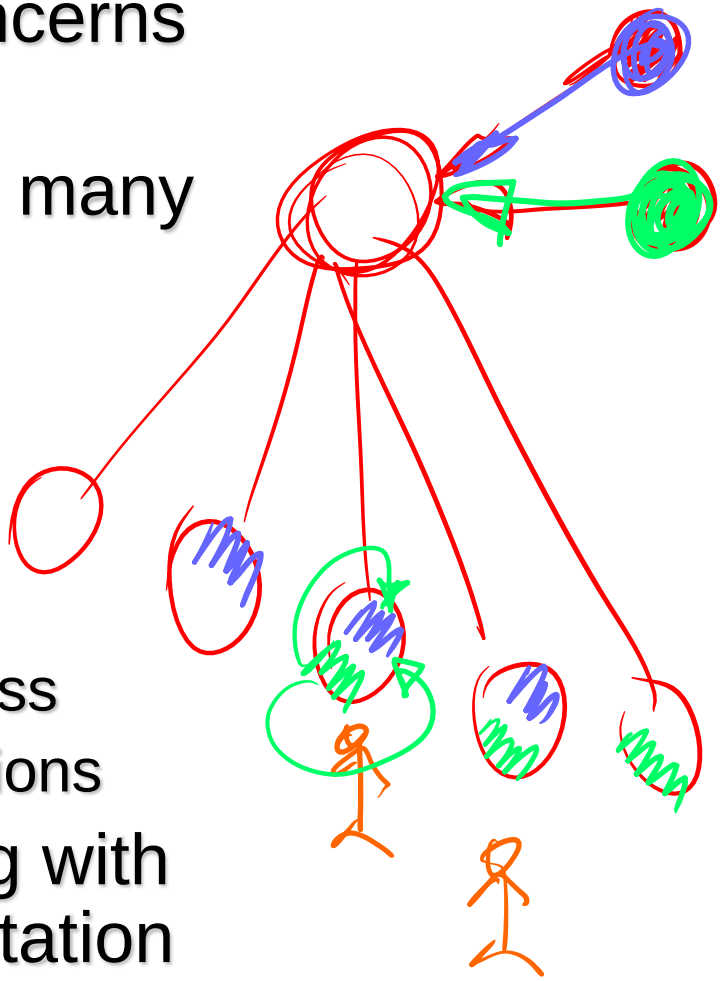
Types of Grid Computing

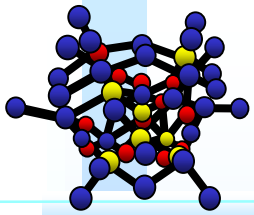




Security Problems

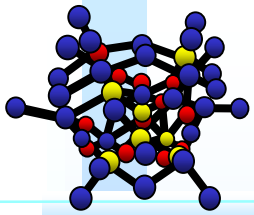
- Address security and policy concerns of resource owners and users
- Are flexible enough to deal with many resource types and sharing modalities
- Scale to
  - large number of resources,
  - many participant/users
  - many program components/process
  - On different nodes and configurations
- Operate efficiently when dealing with large amounts of data & computation



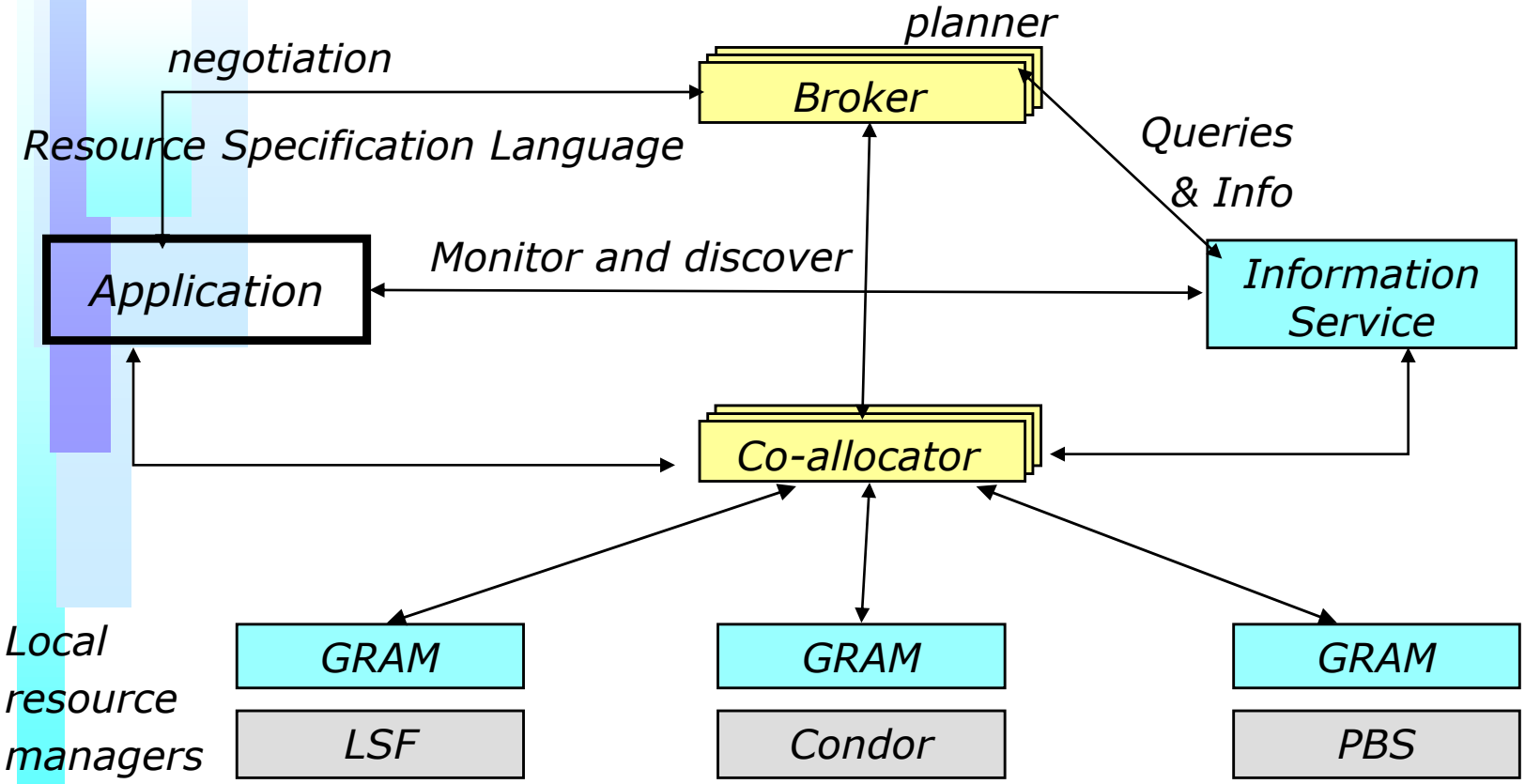


GRID projects

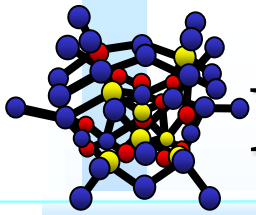
- **GLOBUS**
  - Parallel, Unix like
  - C and java, Open source (era)
- **Condor**
  - Unix and windows
  - Small scale GRID, non parallelism
- **AXMEDIS, AXCP**
  - C++ and JavaScript
  - Windows
  - Accessible Code, Free Software



Resource Management Architecture



- Load Sharing Facility
- Portable Batch System"

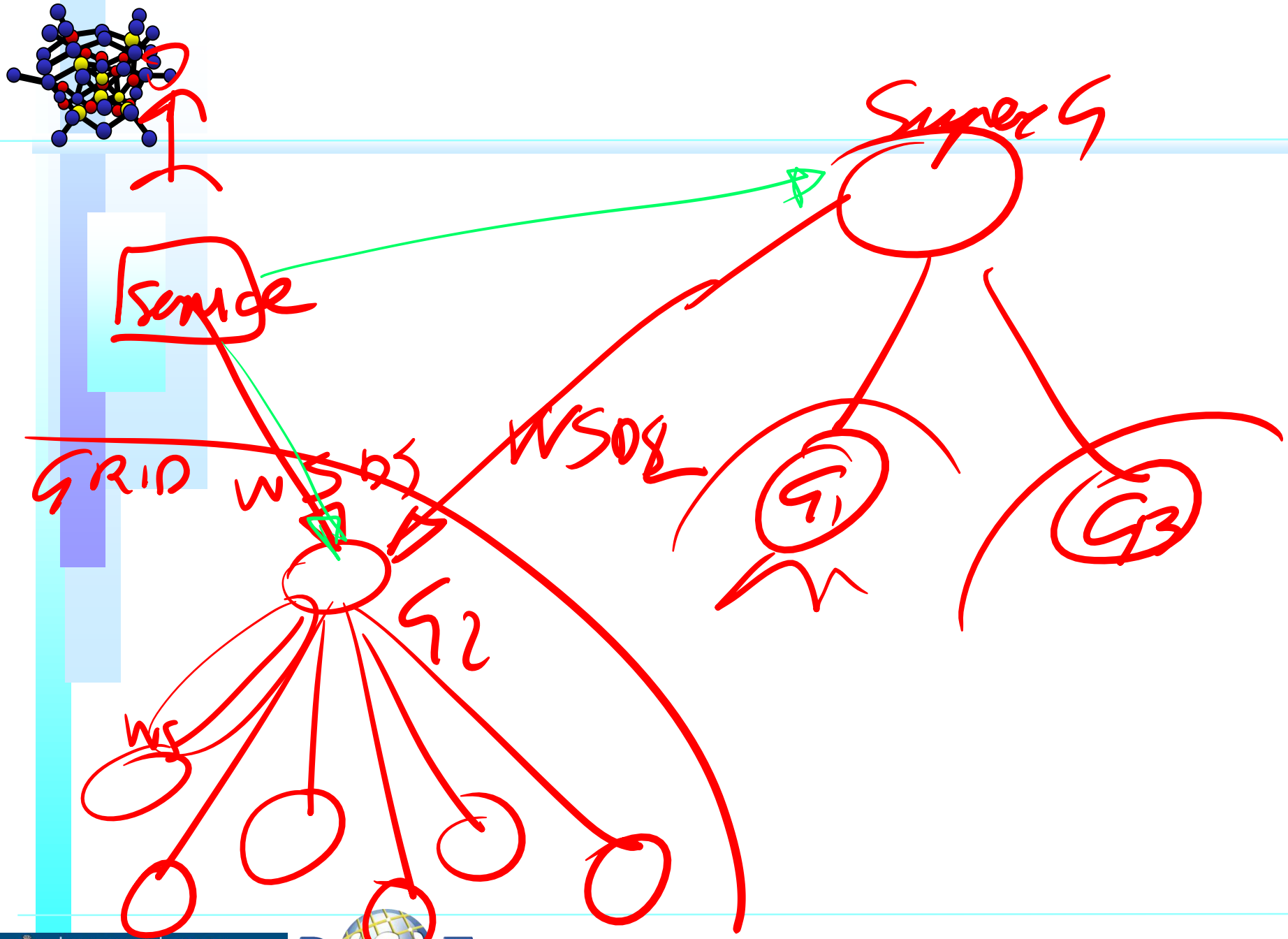


Resource Management Services

the globus project™ www.globus.org

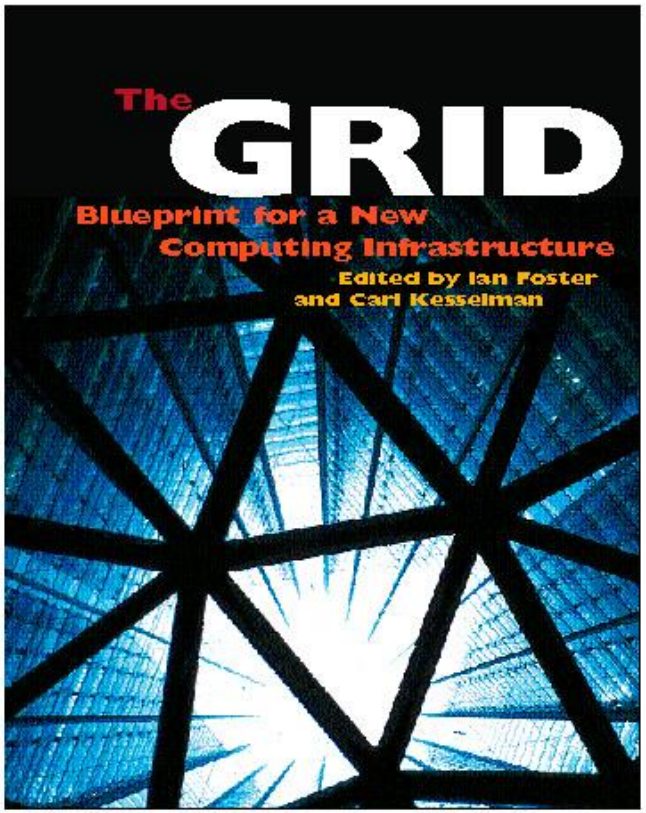
Tre componenti principali

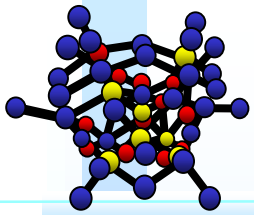
1. **RSL (Resource Specification Language**) per comunicare i requisiti delle risorse
2. **Resource Broker**: gestisce il mapping tra le richieste ad alto livello delle applicazioni e le risorse disponibili.
3. **GRAM** (Grid Resource Allocation Management) è responsabile di un set di risorse ed agisce da interfaccia verso vari Resource Management Tools (Condor,LSF,PBS, NQE, EASY-LL ma anche, semplicemente, un demone per la *fork*)


Super G
Service
GRID
WSRF
WSDL
G1
G3
G2
WS

For More Information

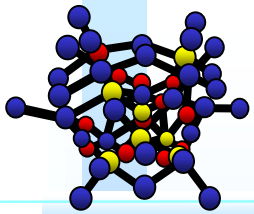
- Globus Project™
  - www.globus.org
- Grid Forum
  - www.gridforum.org
- Book (Morgan Kaufman)
  - www.mkp.com/grids
- Survey + Articoli
  - www.mcs.anl.gov/~foster





sommario

- Contesto tecnologico
- Overhead e Speed-UP
- Aspetti Topologici per il calcolo parallelo
- Soluzioni GRID
- Soluzioni MicroGRID
- Apache Hadoop MapReduce
  - Vedere anche esercitazioni
  - http://www.disit.org/7073
- Apache Hadoop MapReduce 2
- Apache Spark



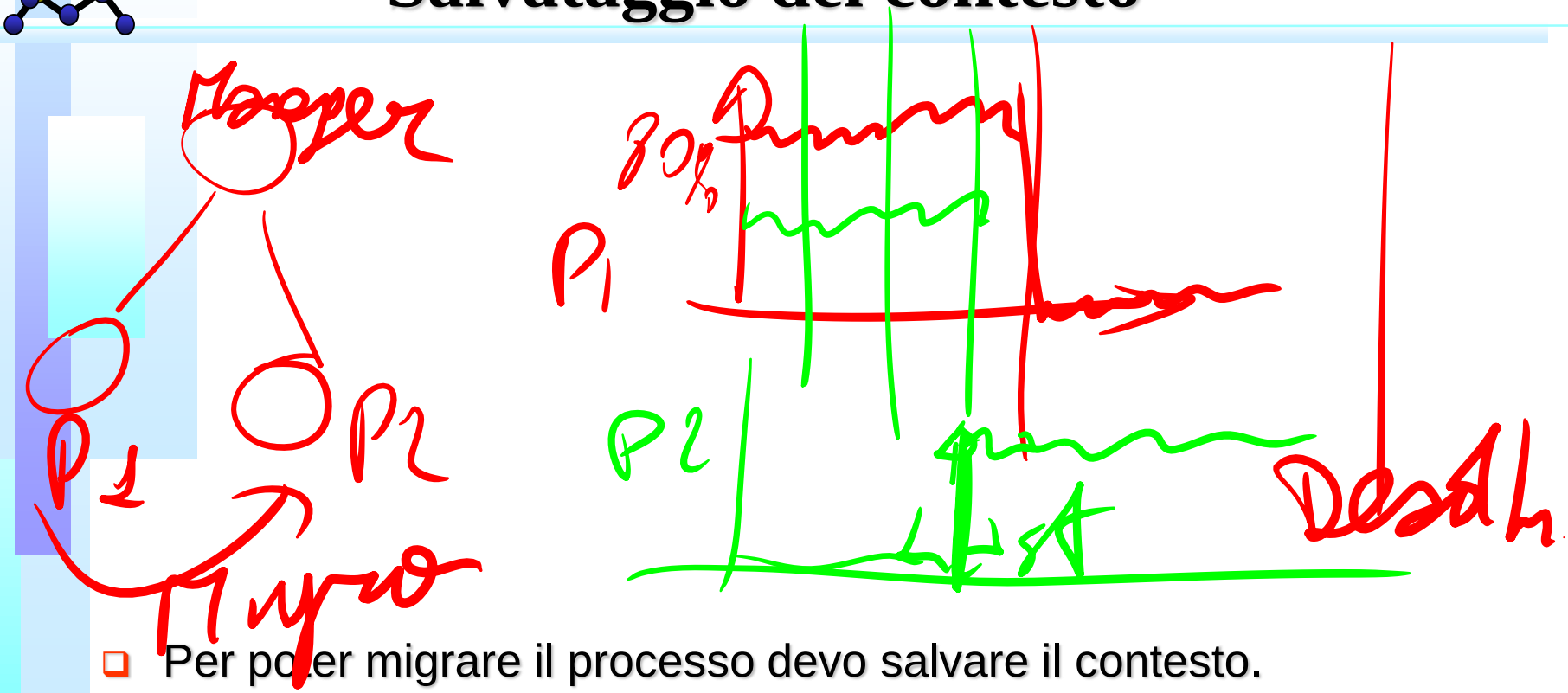
CONDOR

- Nasce nel 1988, University of Madison
- Creazione di cluster di workstation PC

- Sfruttamento di momenti di scarsa attivita' della CPU;
  - Condor lascia la CPU se l'utente lavora sul PC
  - Salva il punto e poi riparte
  - ***Sposta/migra se necessario l'esecuzione del processo su un'altra CPU***

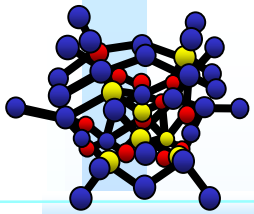
- Il codice non viene cambiato ma viene semplicemente linkato con lib speciali

Salvataggio del contesto



- Per poter migrare il processo devo salvare il contesto.
- Il contesto lo salvo ad intervalli regolari, per esempio ogni decimo del tempo di esecuzione.
- in questo caso ho uno spreco massimo di 1/10 del tempo di esecuzione, che deve essere vantaggioso rispetto al costo di spostamento del processo sull'altro nodo

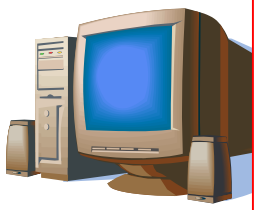
solo nei punti dove è possibile



CONDOR

- A basso livello si basa su procolli di comunicazione diversi per gestire i processi (interoperabilita'):
  - **Vanilla:** permette di eseguire tutti i programmi che non possono essere re-linkati ed è utilizzato per shell scripts. Non sono implementate migrazione e chiamate di sistema.
  - **PVM:** per far girare sopra Condor programmi scritti per l'interfaccia PVM (Parallel Virtual Machine).
  - **MPI:** Questo ambiente risulta utile per eseguire i programmi scritti secondo il paradigma di Message Passing Interface (MPICH).
  - **Globus** Permette di eseguire i processi scritti …..
  - **Java:** Permette di eseguire i processi scritti per la Java Virtual Machine
  - ….

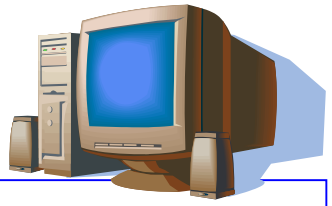
CONDOR architecture



Central Manager

Condor_Collector

Condor_Negotiator



Execution Machine

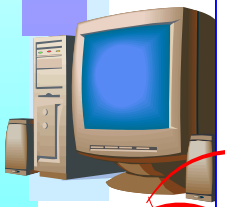
Controlling Daemons

Control via Unix signals to alert job when to checkpoint

User's job

User's code

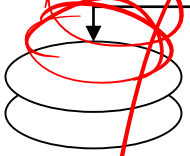
Condor_Syscall_Library



Submit Machine

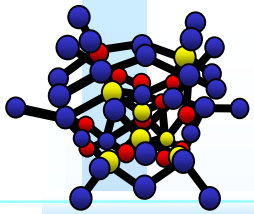
Controlling Daemons

Condor_Shadow Process



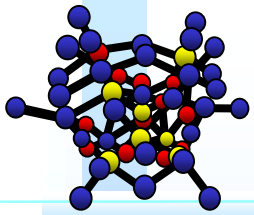
Checkpoint File is Saved to Disk

All System Calls Performed As Remote Procedure Calls back to the Submit Machine



CONDOR ruoli e servizi

- **Central Manager,** solo un Central Manager.
  - Raccoglie tutte le informazioni e negoziare tra le richieste e le offerte di risorse.
  - Affidabile e potente
- **Submit**
  - Altre macchine del pool (incluso il Central Manager) possono invece essere configurate per sottomettere jobs a Condor.
  - queste macchine necessitano di molto spazio libero su disco per salvare tutti i punti di checkpoint dei vari job sottomessi.
- **Execute (i nodi del GRID)**
  - Alcune macchine nel pool (incluso il Central Manager) possono essere configurate per eseguire processi Condor.
  - Essere una macchina di esecuzione non implica la richiesta di molte risorse.

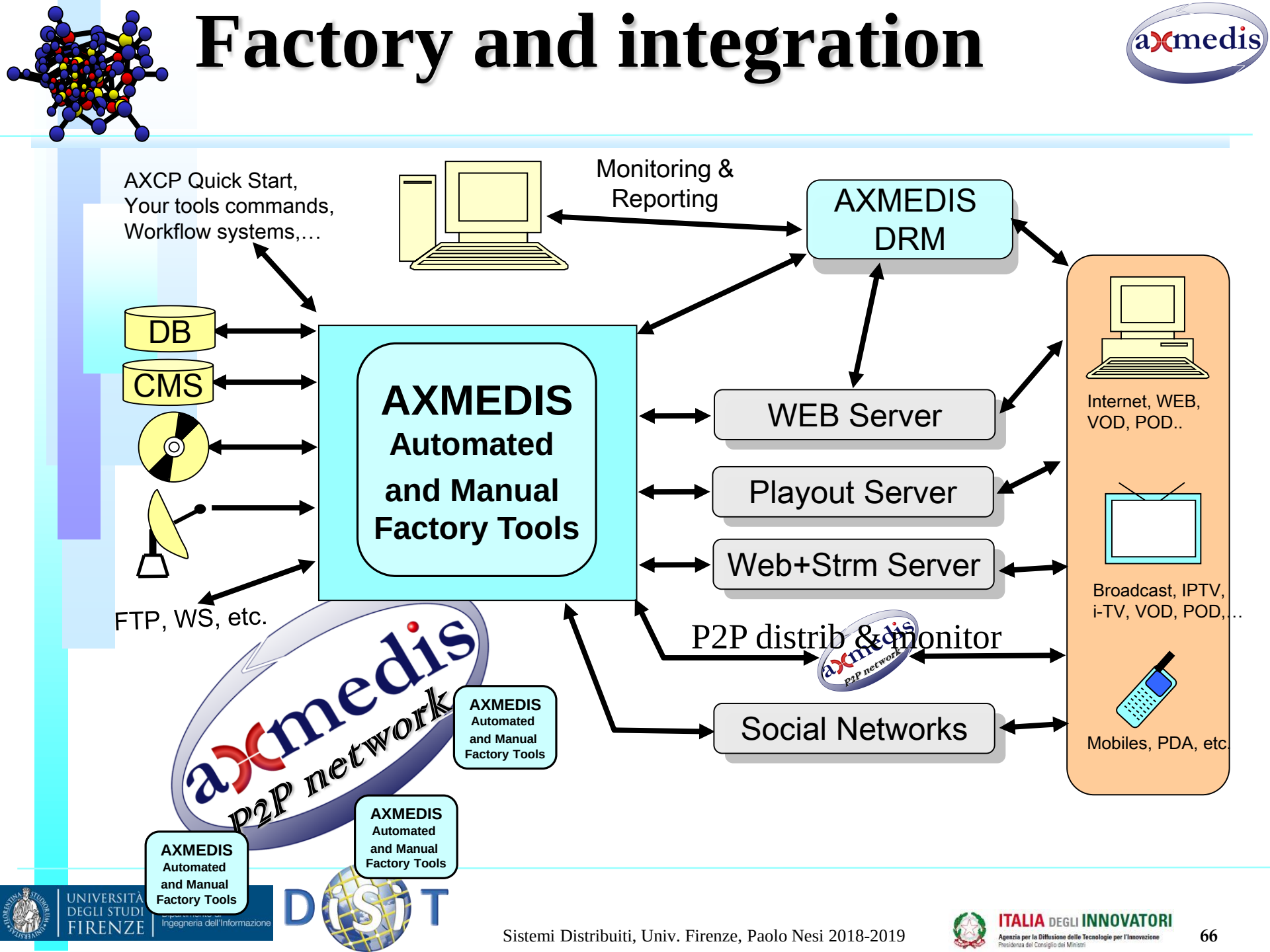


CONDOR: Pregi e difetti

- ***Pregi:***
  - non necessita di modificare i vostri programmi
    - ➔Differentemente da seti@home
  - set-up semplice
  - facilità di gestione della coda dei job
  - breve tempo necessario ad implementare una *"griglia"* funzionante.
  - estrema versatilità nel gestire svariati tipi di applicazioni (.exe).
  - trasparenza agli occhi degli utenti durante l'esecuzione.

- ***Difetti:***
  - I meccanismi di sicurezza implementati non garantiscono ancora il medesimo livello di protezione offerto da una vera soluzione *middleware*.
  - Limitato controllo sull'intera *grid*.
  - Bassa "tolleranza" ai guasti: se nessuna macchina del pool soddisfa i requisiti di un job, questo rimane in attesa senza andar mai in esecuzione e l'utente è costretto a rimuoverlo manualmente.

Factory and integration



AXMEDIS Content Processing GRID

Front end servers, VOD, prod on demand

Workflow manager

AXCP Visual Designer

AXMEDIS Rule Editor

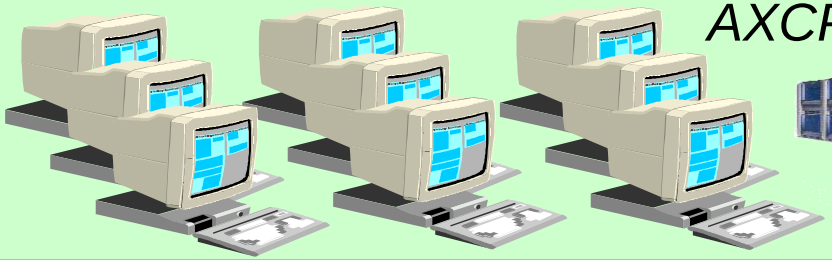
Visual Elements and Rules

Quick Starter

AXCP Scheduler

AXCP GRID Rules

AXCP nodes



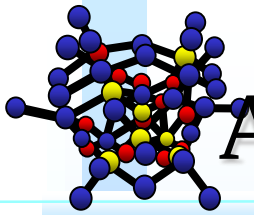
Plug-in for content processing

WS, FTP, etc.

Your CMSs

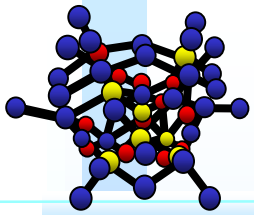
AXMEDIS Database

DistributionChannels and servers



AXMEDIS Content Processing GRID

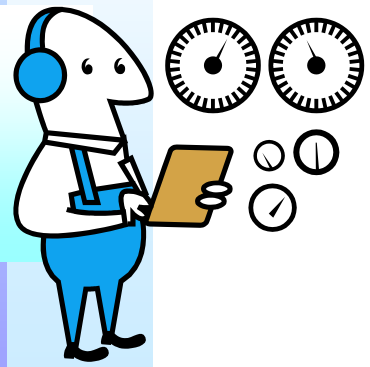
- **GRID per il Content Processing**
  - Discovery di risorse, nodi
    - Valutazione dello stato e delle pontenzialita dei nodi
  - Creazione di regole, processi
    - Un solo processo per nodo
  - Esecuzione di regole/processi, attivano anche processi locali scritti non in forma di regole
    - On demand, periodiche, collegate, asincrone
  - Allocazione ed ottimizzazione dei processi
  - Comunicazione con il gestore ma anche fra nodi
    - N to N
    - N to S, per monitor e/o invocazione di regole/processi
  - Tracciamento e controllo dei processi
  - Workflow, gestione di alto livello, integrazione macchina Users



AXCP processing capabilities

- **Processing functionalities:**
  - Gathering content
  - Production of new objects: composition, etc.
  - Formatting: SMIL, XSLT, etc.
  - Synchronization of media, etc.
  - Content adaptation, transcoding, …….
  - Reasoning on device capabilities and user preferences, (user, device, network, context)
  - Production of licenses, MPEG-21 REL, OMA
  - Verification of Licenses against them and PAR
  - Metadata and metadata mapping: Dublin Core, XML
  - Extraction of descriptors and fingerprints …MPEG-7
  - Protocols: IMAP, POP, Z39.50, WebDav, HTTP, FTP, WS, etc.
  - Controlling P2P networks
  - ....
- **Any type of resource** in any format
  - Multimedia: MPEG21, IMS, SCORM, etc.
  - DB: ODBC, JDBC, DB2, Oracle, MS-SQL, MySQL, XML databases, etc.
  - Licensing systems: MPEG-21, OMA
  - File Formats: any video, audio, image, xml, smil, html, ccs, xslt, etc.

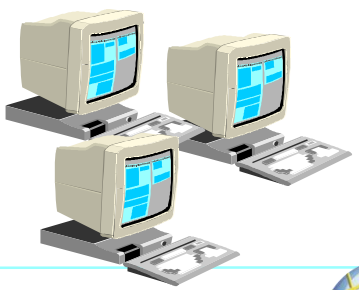
AXMEDIS Content Processing GRID

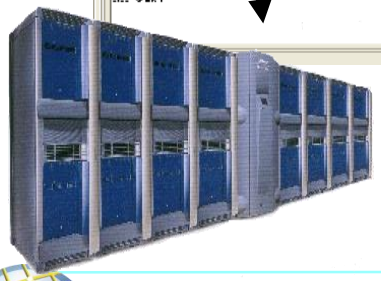


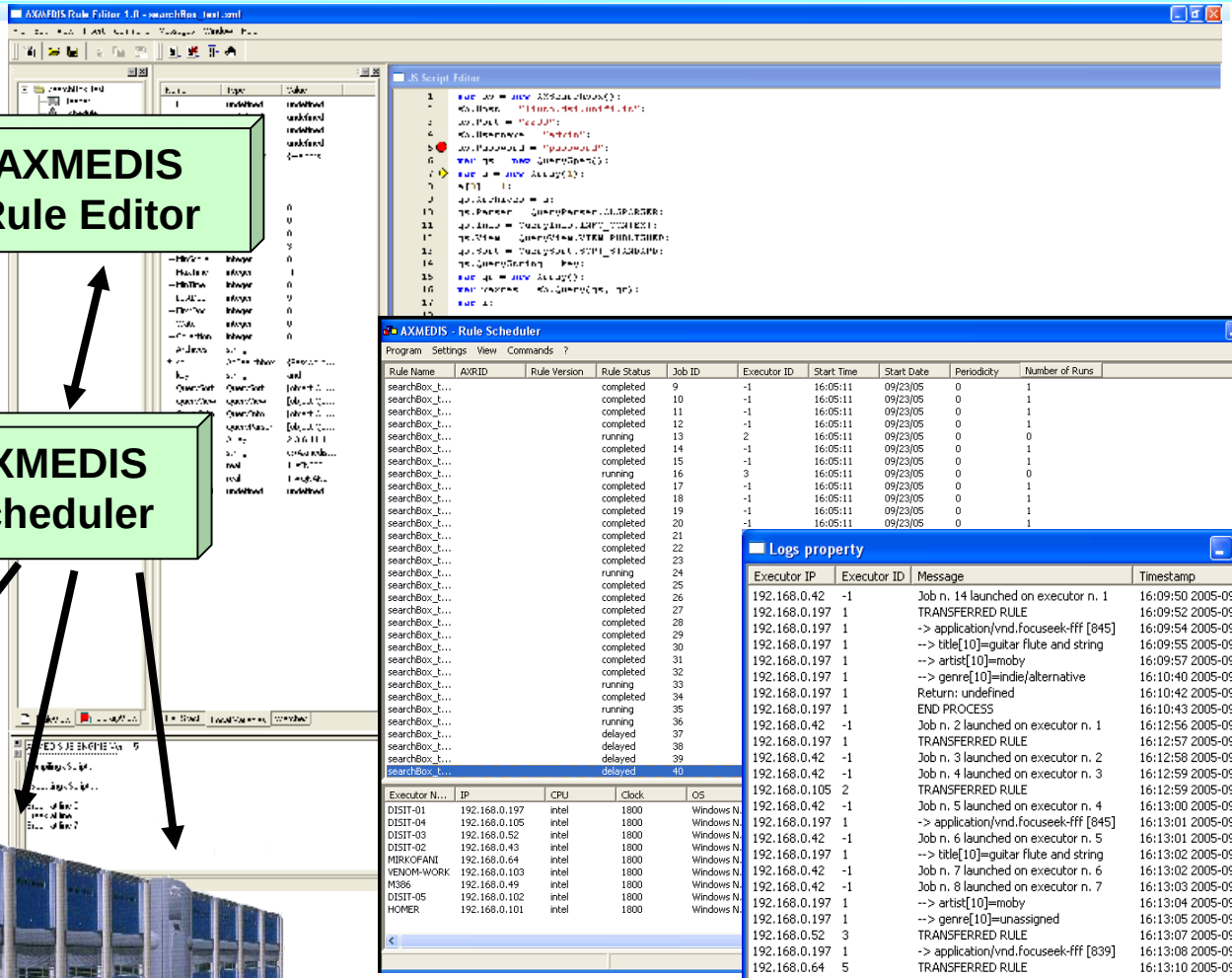
AXMEDIS Rule Editor

AXMEDIS Scheduler

Workflow manager

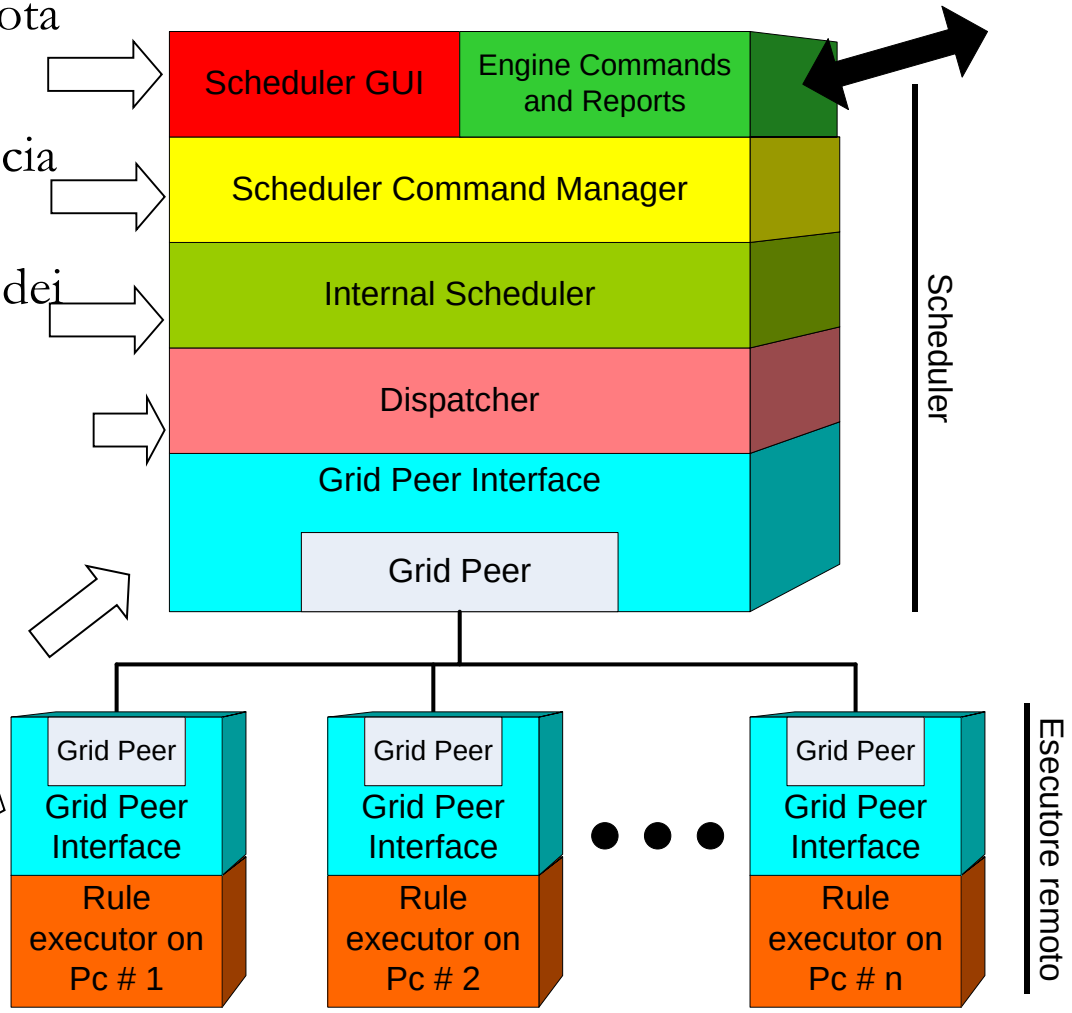






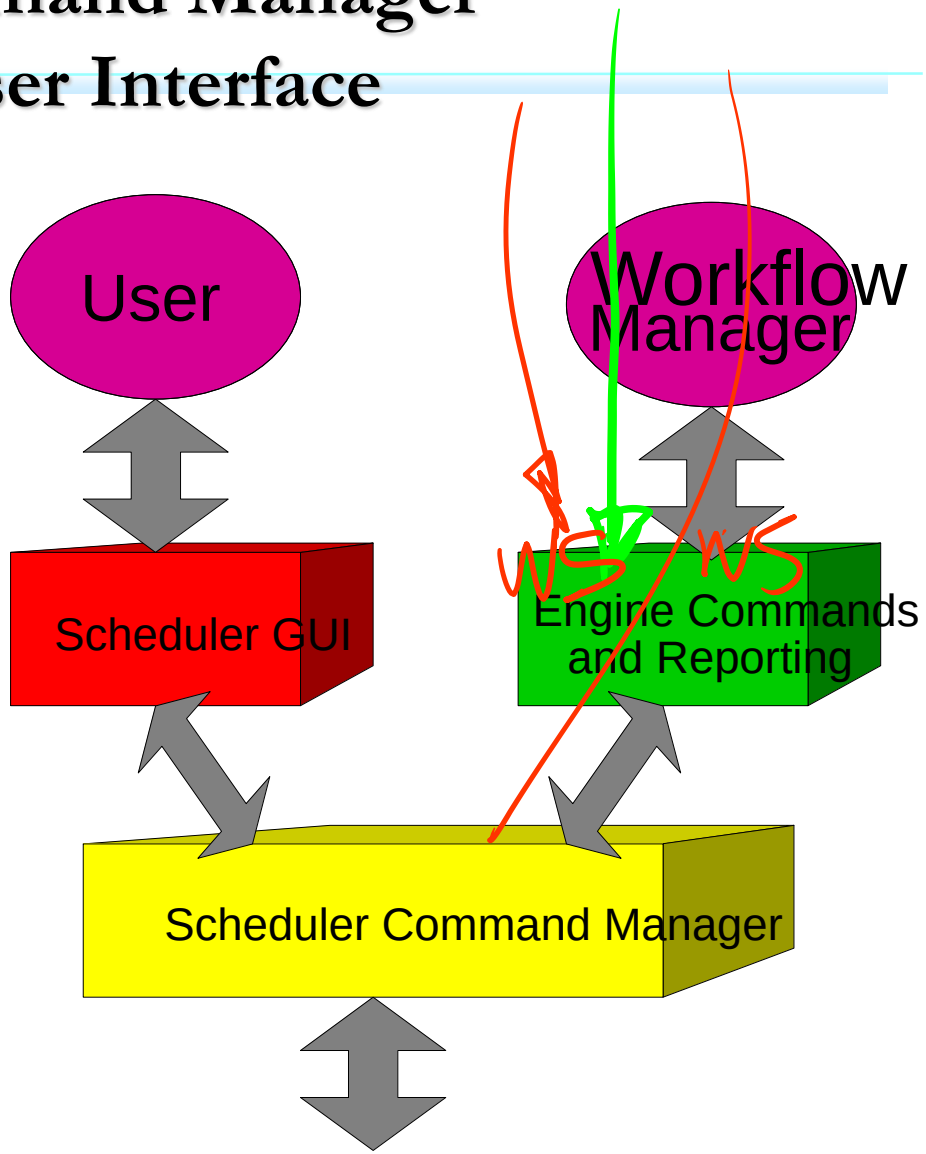
Realizzazione: Rule Engine

- **Eng. Cmd & Rep.:** interfaccia remota verso il workflow manager
- **Scheduler Cmd Manager**: interfaccia dello scheduler
- **Internal Scheduler**: schedulazione dei job, supervisione del dispatcher
- **Dispatcher**: gestore delle risorse remote, della comunicazione e dell'associazione job-esecutore
- **Grid Peer Interface**: modulo per la gestione della comunicazione remota
- **Rule Executor**: modulo per l'esecuzione dello script

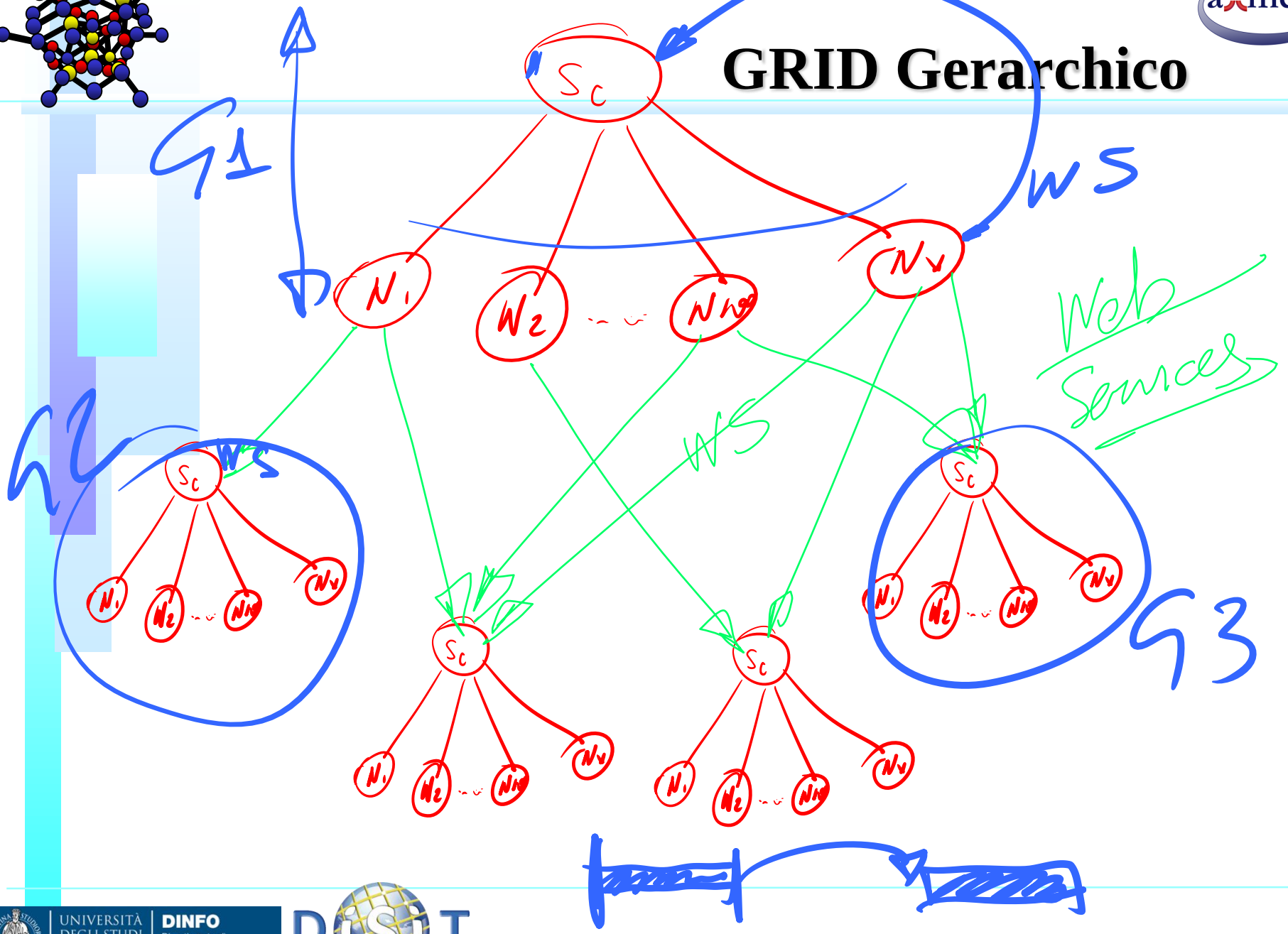


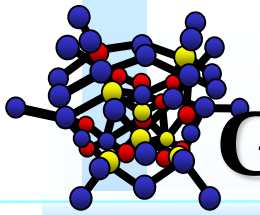
Scheduler Command Manager e Graphic User Interface

- Lo Scheduler Command Manager è un'interfaccia che rende indipendente lo scheduler dal tipo di interfaccia utente (grafica o di comunicazione remota) usata
- L'interfaccia grafica permette un accesso rapido alle funzionalità dell'applicazione



GRID Gerarchico



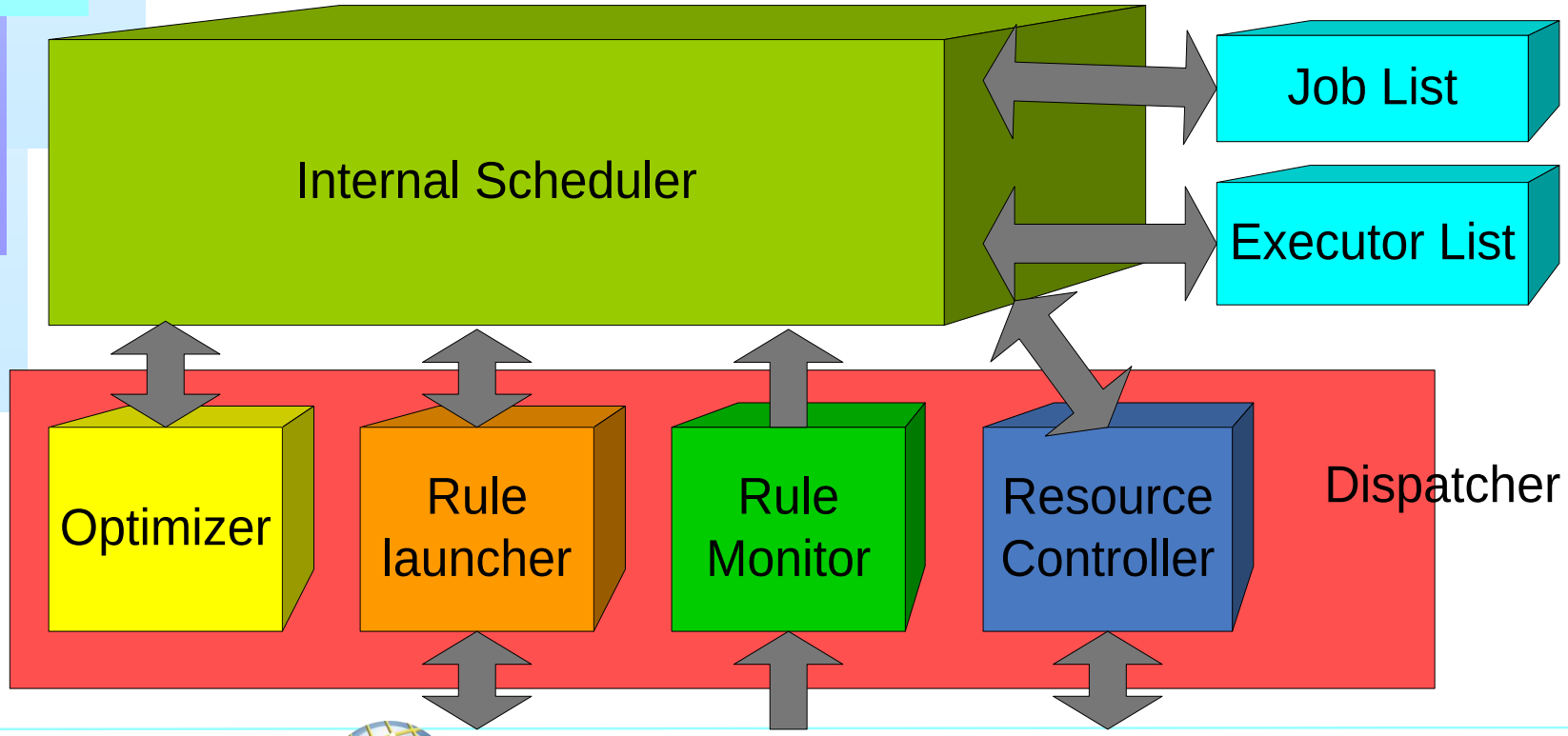


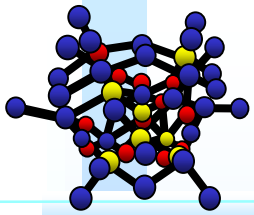
Gestione Gerarchica di microGRID

- Ogni Scheduler identifica un microGRID
- Da ogni nodo del GRID e' possibile inviare richieste ad altri Scheduler e pertanto ad altri microGRID
  - Le richieste vengono inviate tramite chiamate a Web Services
- Si viene a greare una gerarchia di grid e di nodi in tali grid
- I singoli MicroGRID possono essere distirbuiti anche geograficamente, si veine a creare un vero e proprio GRID geografico
- I nodi foglia possono inviare richieste allo scheduler radice, etc.

Internal Scheduler

- permette la scelta dei job da mandare in esecuzione e l'aggiornamento della periodicità, il controllo sulla scadenza, il controllo e l'aggiornamento delle liste dei job e degli esecutori

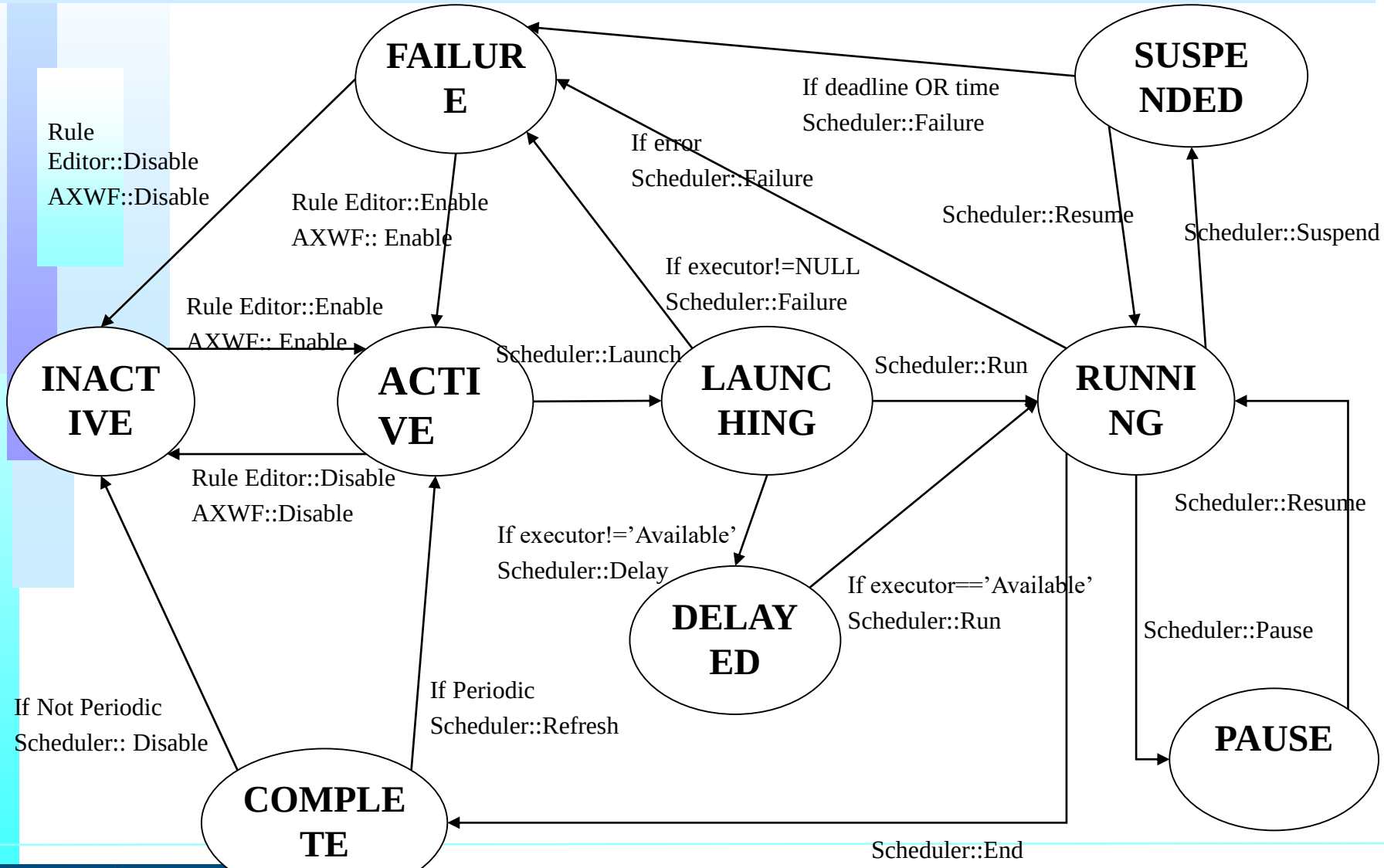




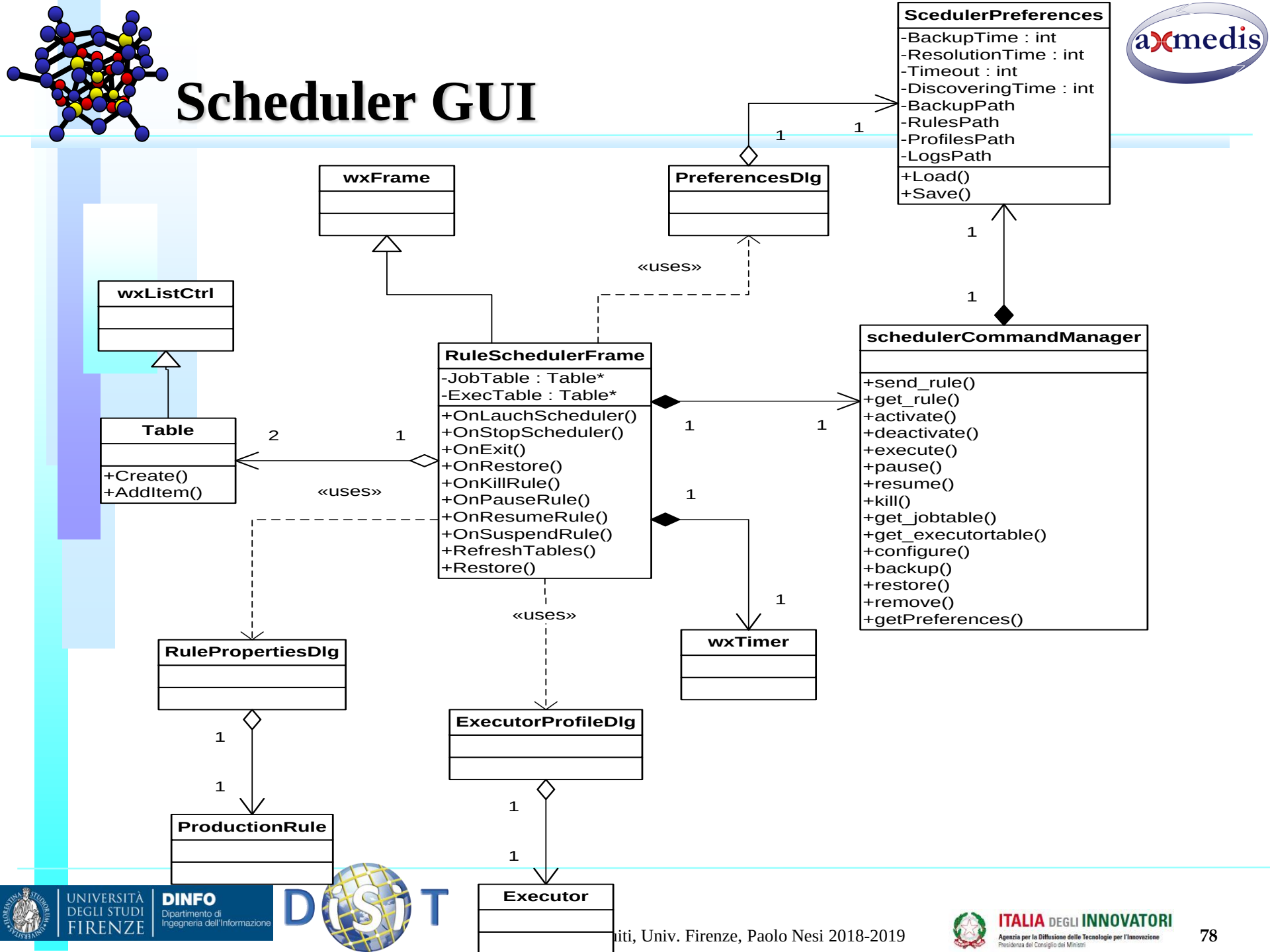
Dispatcher

- riunisce le funzionalità di associazione, lancio e controllo:
  - **Resource Controller:** controllo dello stato degli esecutori e la richiesta del loro profilo
  - **Optimizer:** associazione degli esecutori con i job da mandare in esecuzione in funzione del profilo e politica FCFS
  - **Rule Launcher:** invio di comandi e del file del job agli esecutori remoti
  - **Rule Monitor:** controllo sulle notifiche inviate dagli esecutori e dai job in esecuzione

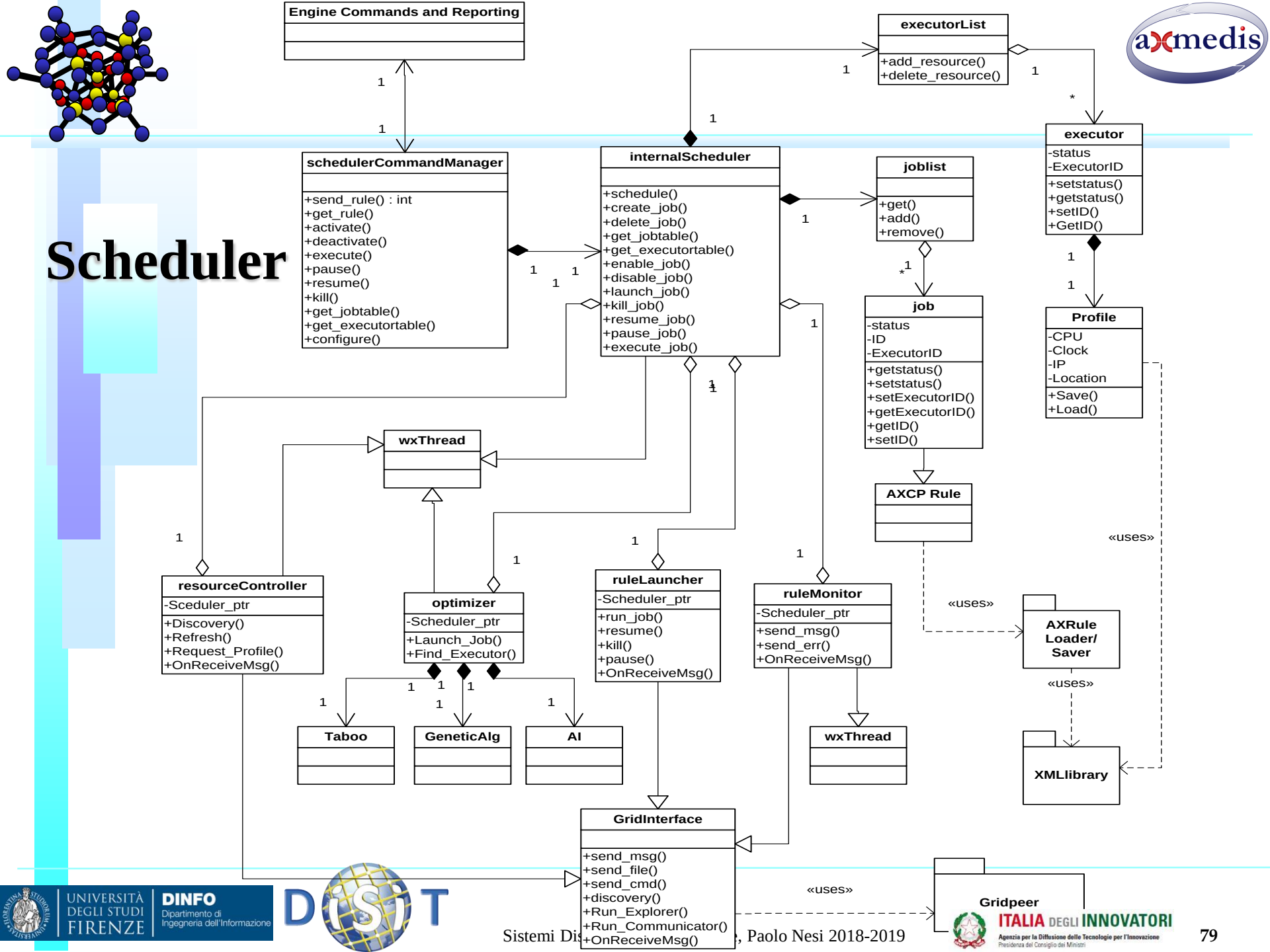
Evolution of Rule's State

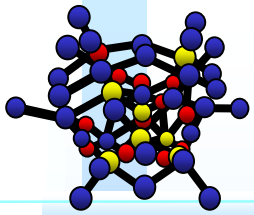


Scheduler GUI

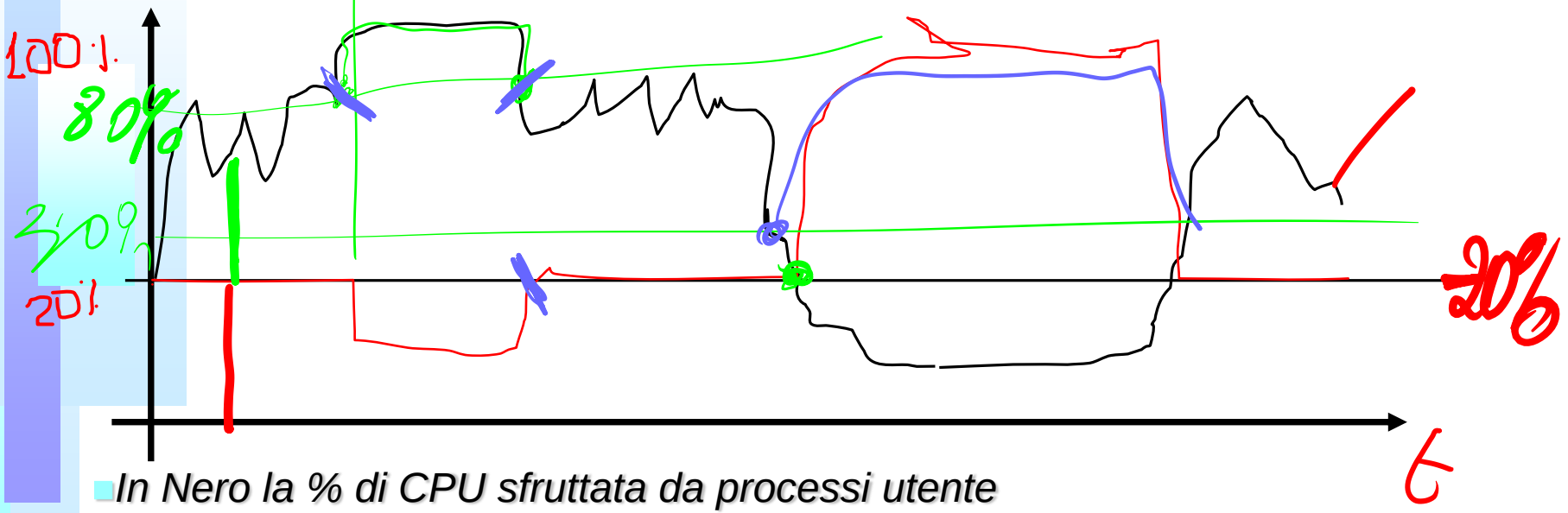


Scheduler


Engine Commands and Reporting
executorList
+add_resource()
+delete_resource()
schedulerCommandManager
+send_rule() : int
+get_rule()
+activate()
+deactivate()
+execute()
+pause()
+resume()
+kill()
+get_jobtable()
+get_executortable()
+configure()
internalScheduler
+schedule()
+create_job()
+delete_job()
+get_jobtable()
+get_executortable()
+enable_job()
+disable_job()
+launch_job()
+kill_job()
+resume_job()
+pause_job()
+execute_job()
joblist
+get()
+add()
+remove()
executor
-status
-ExecutorID
+setstatus()
+getstatus()
+setID()
+GetID()
job
-status
-ID
-ExecutorID
+getstatus()
+setstatus()
+setExecutorID()
+getExecutorID()
+getID()
+setID()
Profile
-CPU
-Clock
-IP
-Location
+Save()
+Load()
wxThread
AXCP Rule
«uses»
resourceController
-Sceduler_ptr
+Discovery()
+Refresh()
+Request_Profile()
+OnReceiveMsg()
optimizer
-Scheduler_ptr
+Launch_Job()
+Find_Executor()
ruleLauncher
-Scheduler_ptr
+run_job()
+resume()
+kill()
+pause()
+OnReceiveMsg()
ruleMonitor
-Scheduler_ptr
+send_msg()
+send_err()
+OnReceiveMsg()
AXRule Loader/ Saver
Taboo
GeneticAlg
AI
wxThread
XMLlibrary
GridInterface
+send_msg()
+send_file()
+send_cmd()
+discovery()
+Run_Explorer()
+Run_Communicator()
+OnReceiveMsg()
Gridpeer

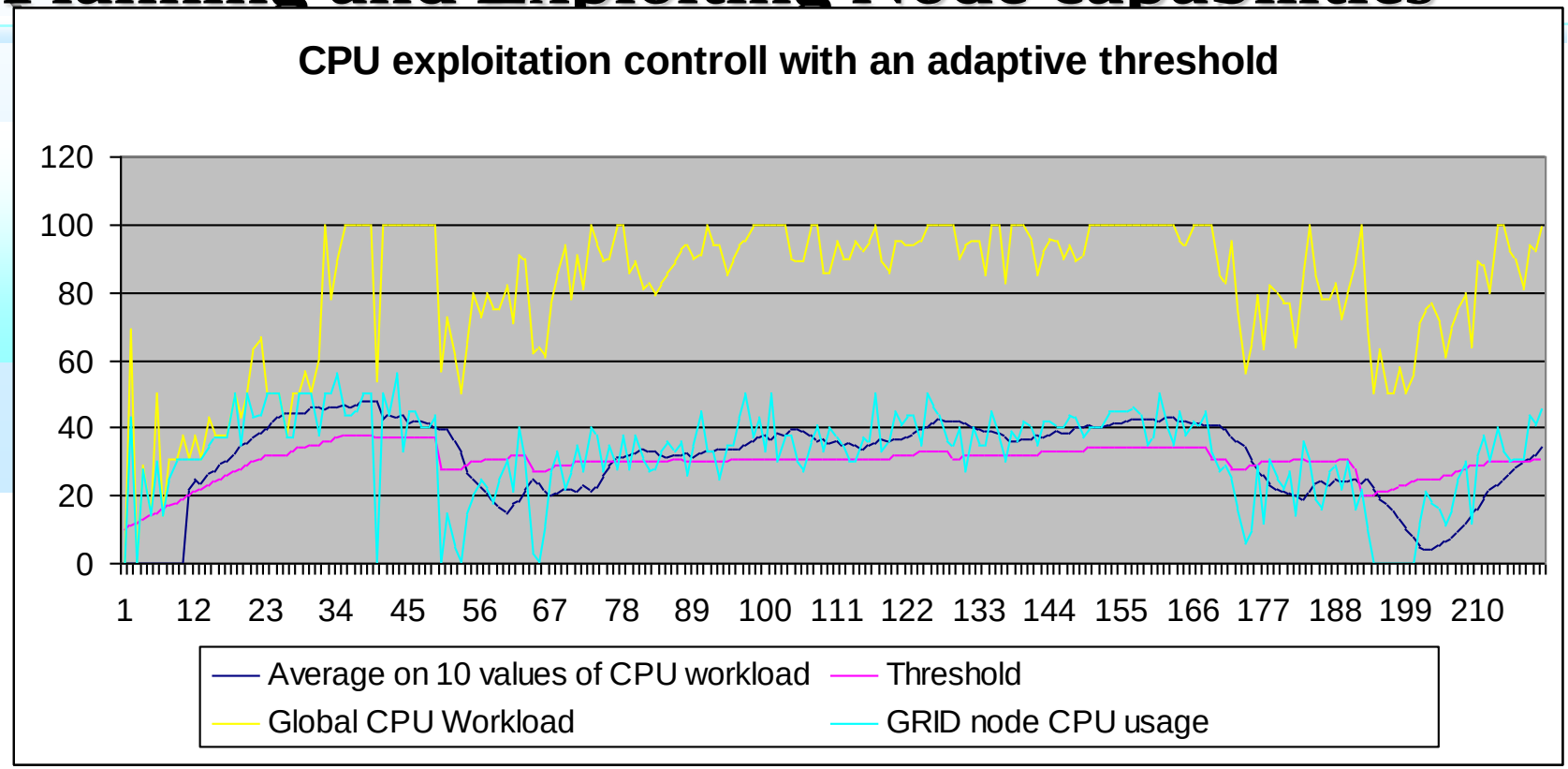


Sfruttamento della CPU nei nodi

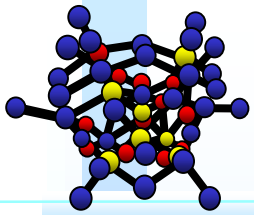


- *In Nero la % di CPU sfruttata da processi utente*
- *In Rosso la % di CPU sfruttata dal processo del GRID*
- *Politiche*
  - *Stare strettamente sotto/allo X%*
  - *Accettare che si possa andare sotto, ma stare all'X% come valore medio*
  - *Sfruttare al massimo la CPU quando l'utente non e' presente*

Planning and Exploiting Node capabilities



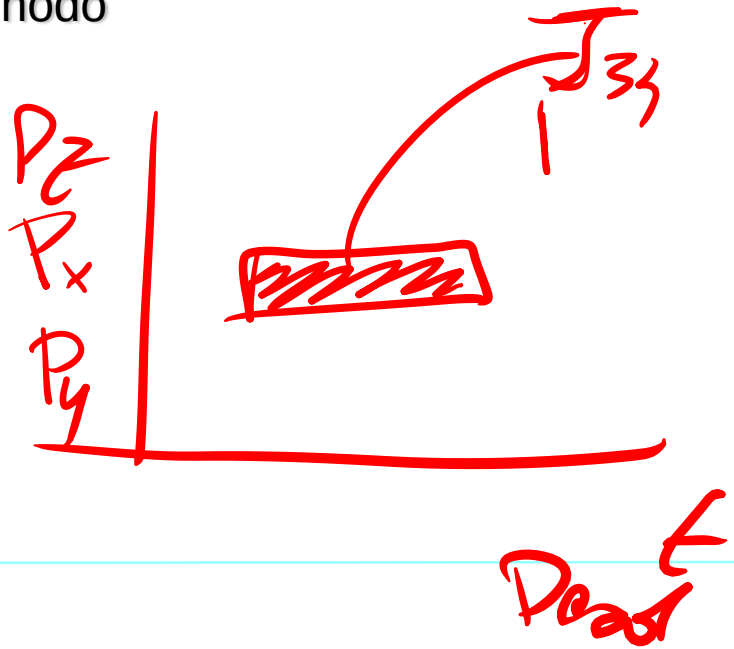
- GRID node has a profile describing its capabilities: time profile, memory, HD, communication, tools and plug ins, etc.

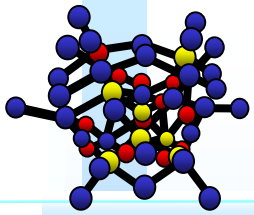


Ottimizzazione della Pianificazione

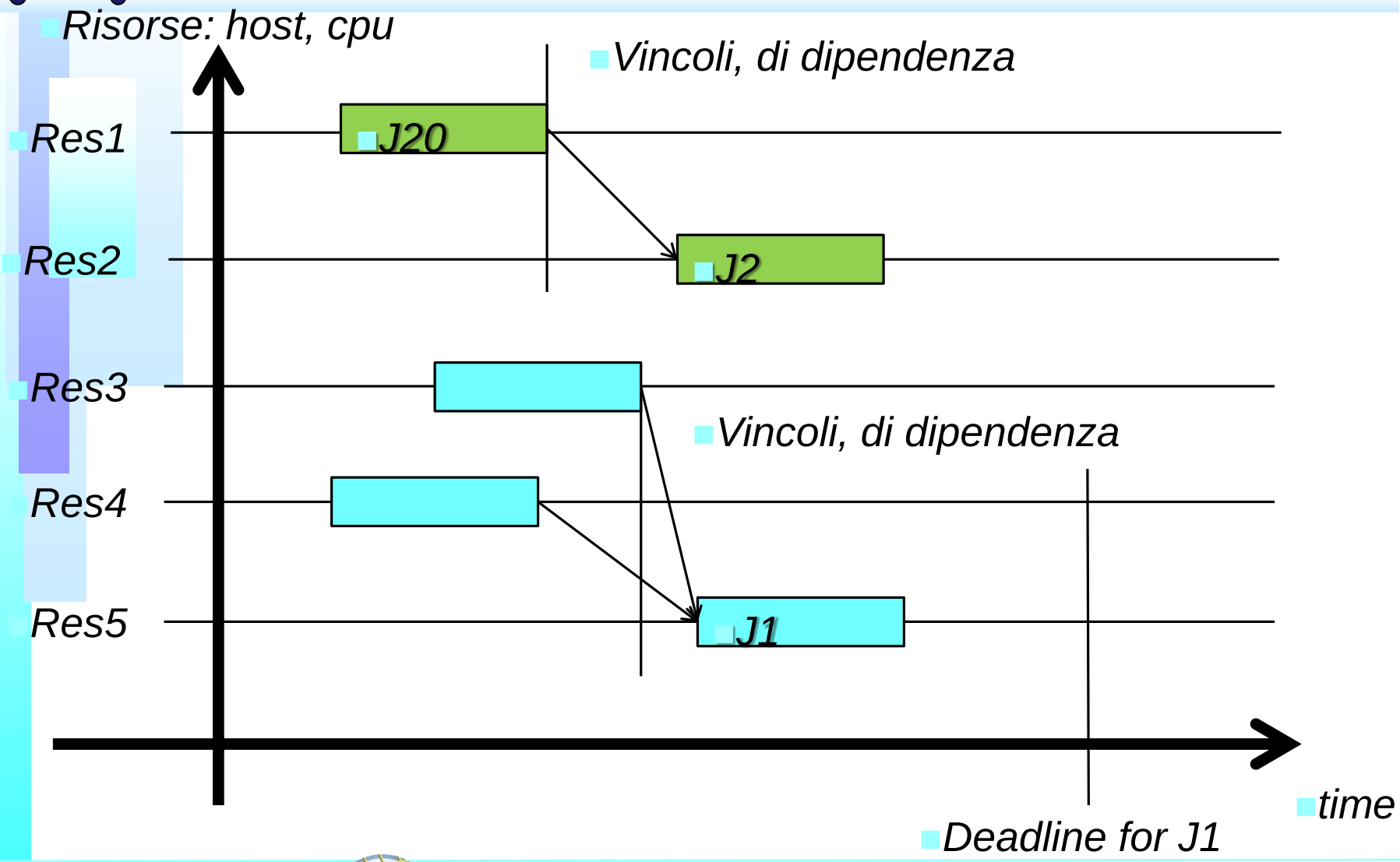
Allocazione dei processi sui nodi

- Valutazione del profilo dei Nodi
  - Capabilities dei nodi
  - Potenza computazionale
  - Network Capabilities
- Valutazione delle necessità delle regole/processi
  - Componenti necessari, funzioni necessarie
  - Scadenza temporale, **deadline**
  - Architettura per la sua esecuzione se multi nodo
- Scelta della soluzione ottima:
  - Bilanciamento del carico
  - Soddisfazione dei vincoli
- Algoritmi di allocazione
  - Deadline monotonic
  - Taboo Search
  - Genetic Algorithms

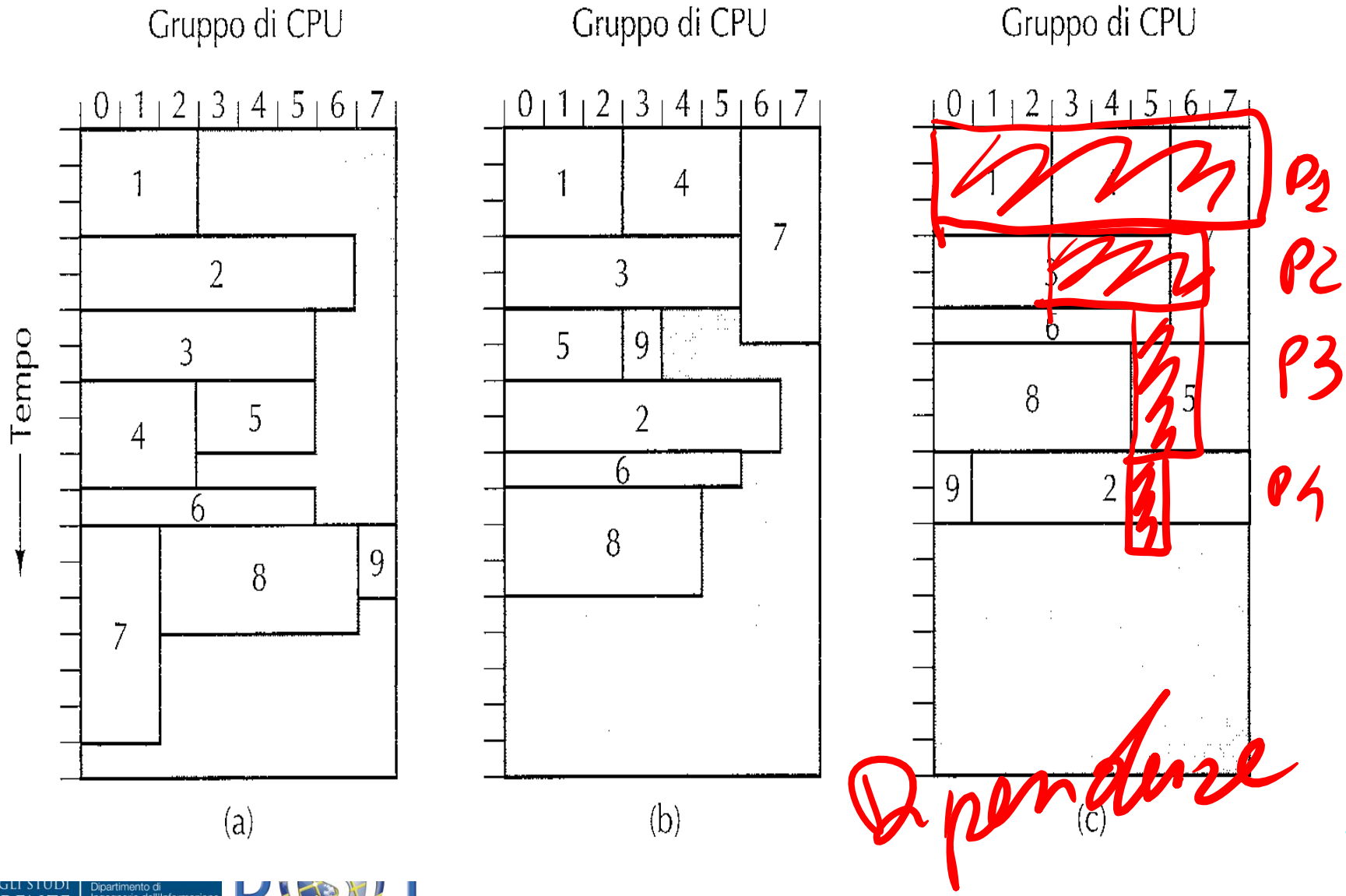


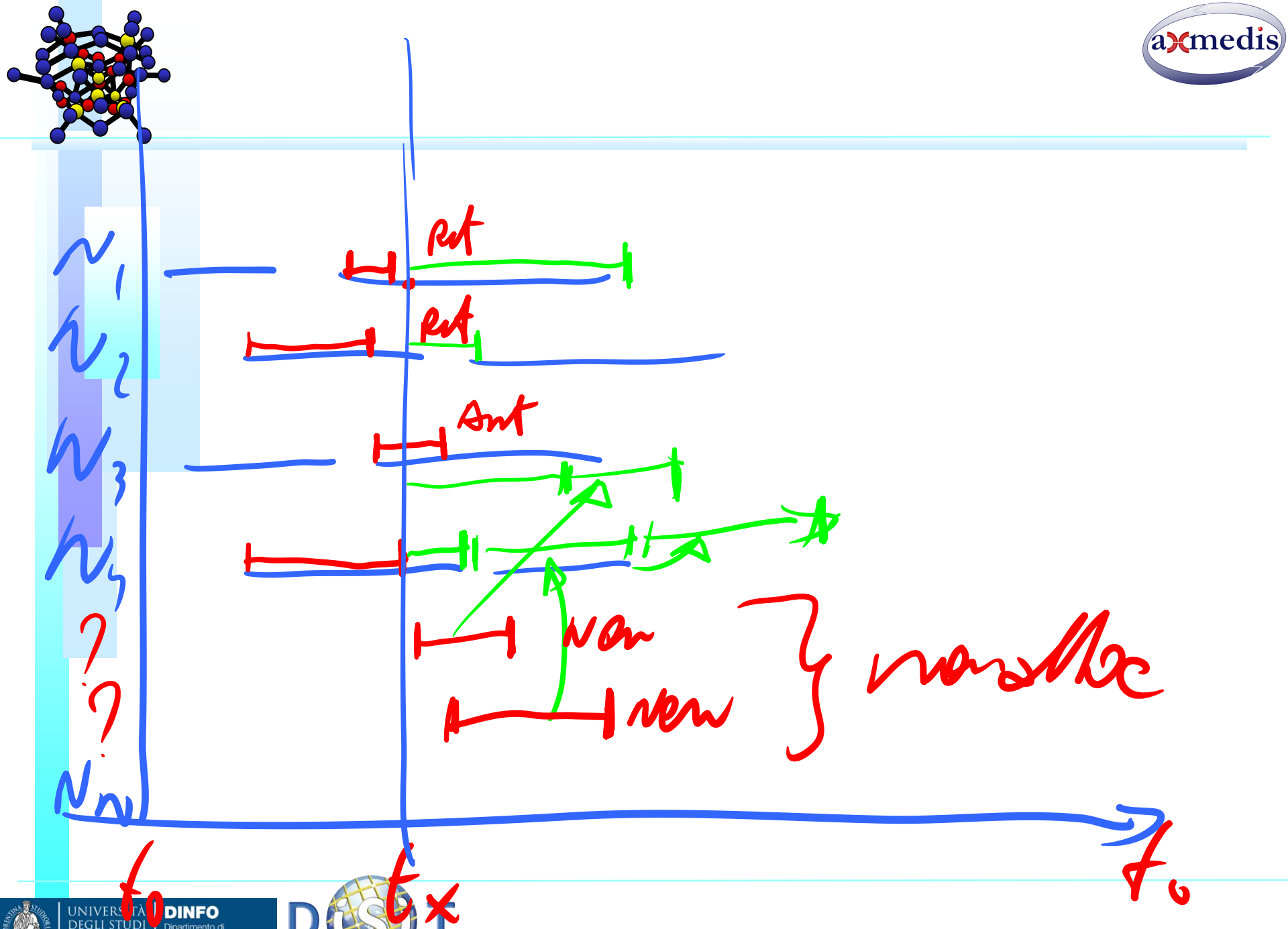


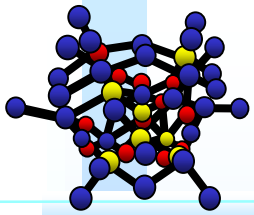
Gantt diagram and jobs



Pianificazione dei processi

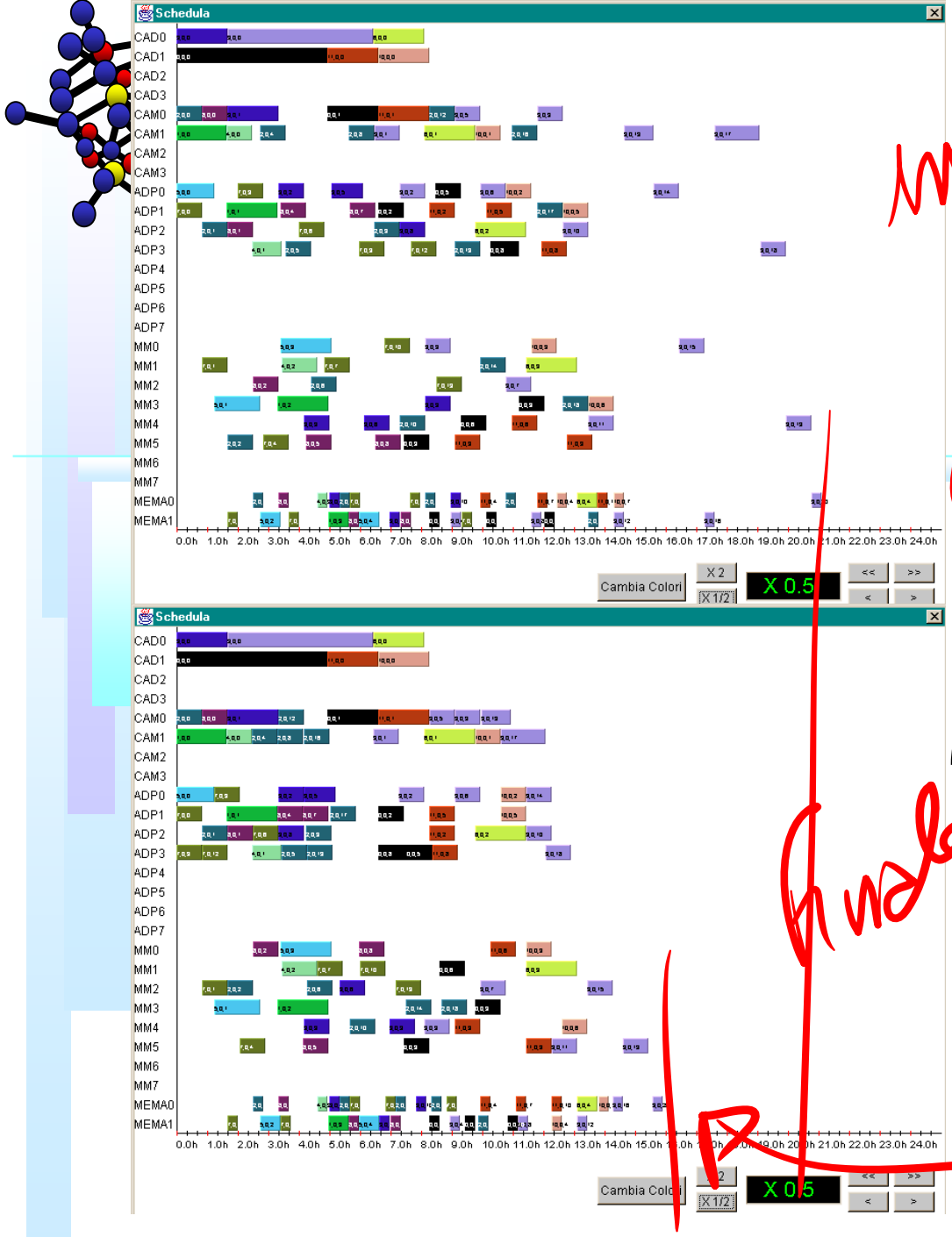



Ref
Ref
Ant
Non
New
t0
t
tf



Per ogni processo al tempo Tx

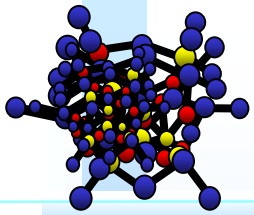
- D: Durata prevista del processo (con incertezza)
- Tstart: Tempo di start (time stamp di start)
- Tend: Tempo previsto di completamento
- Pc: Percentuale di completamento (se possibile), per esempio
  - Valore dell'indice di un processo iterativo,
  - numero di iterazioni, etc. etc.
- Ad un certo Tx>Tstart la Pc puo' essere:
  - Conforme (Tx-Tstart)/D%==Pc
  - In anticipo Pc > (Tx-Tstart)/D%
  - In ritardo Pc < (Tx-Tstart)/D%
- Necessario attualizzare



Esempio di ottimizzazione

125 task, 2CAD, 2CAM, 4ADP, 6MM, 2MEMA

Ottimizzazione: 24.6%



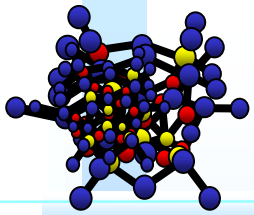
Confronto AG, TS

In letteratura le tecniche più utilizzate per affrontare tali problemi sono le seguenti:

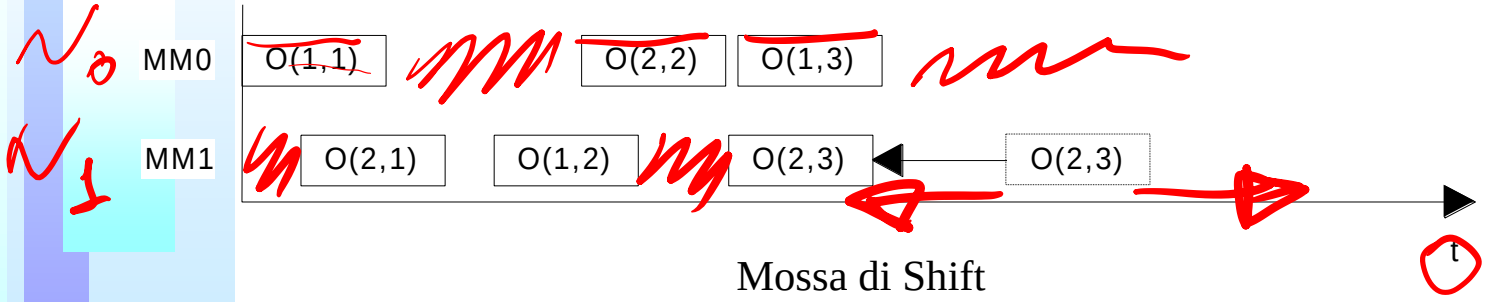
- **AG** (Algoritmi Genetici)
- **TS** (Tabu Search)

Dai lavori utilizzati come riferimento risulta che:

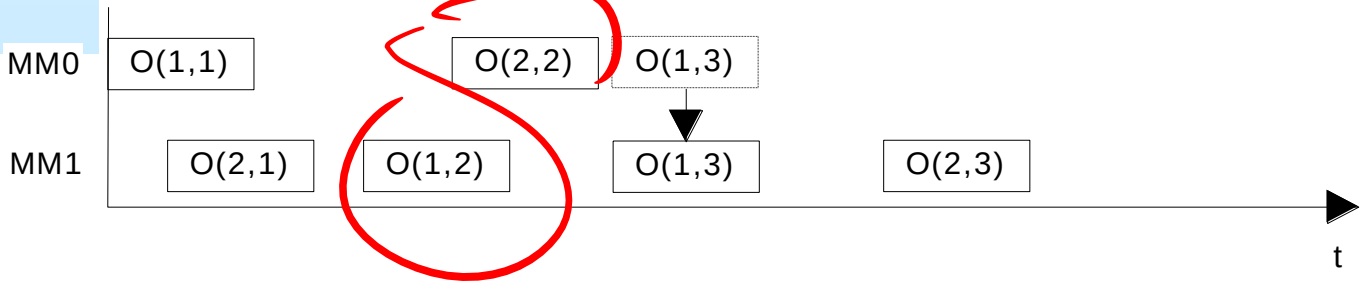
- **TS** maggior velocità (almeno un ordine di grandezza)
- **TS** capacità di trovare il maggior numero di soluzione ottime (benchmark)
- **TS** minor dipendenza rispetto al problema (aumento job e macchine)
- **TS** architettura realizzativa semplice (adatta a vari problemi)
- **AG** maggior robustezza (non necessita di soluzione iniziale). Versioni modificate come **GLS** hanno prodotto buoni risultati.



Mosse operate sui task

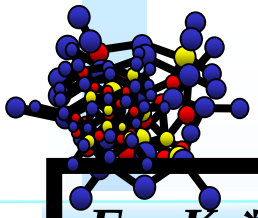


Mossa di Shift



Mossa di Allocazione

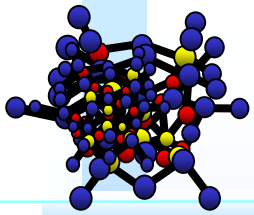
Qualsiasi mossa più complessa può essere ottenuta per composizione di queste mosse semplici



Funzionale di costo

$$F = K_A * allocation - K_B * \Delta biasDeadline + K_D * \Delta delay + K_V * \Delta varTotale + K_c * \Delta C_{max}$$

- allocation costituisce il valor medio di violazione di contemporaneità nella schedula
- C_{max} misura la lunghezza temporale della schedula
- biasDeadline è il valor medio relativo all'anticipo (o al ritardo) rispetto alle scadenze delle operazioni contenute nella schedula
- delay favorisce l'anticipo dei task in ritardo rispetto a quelli che rientrano nella scadenza prevista
- vatTotale costituisce una misura del valor medio del carico complessivo sulle risorse utilizzate
- I vari K sono i pesi associati ai funzionali. I □ indicano che si prende la variazione del funzionale rispetto all'iterazione precedente

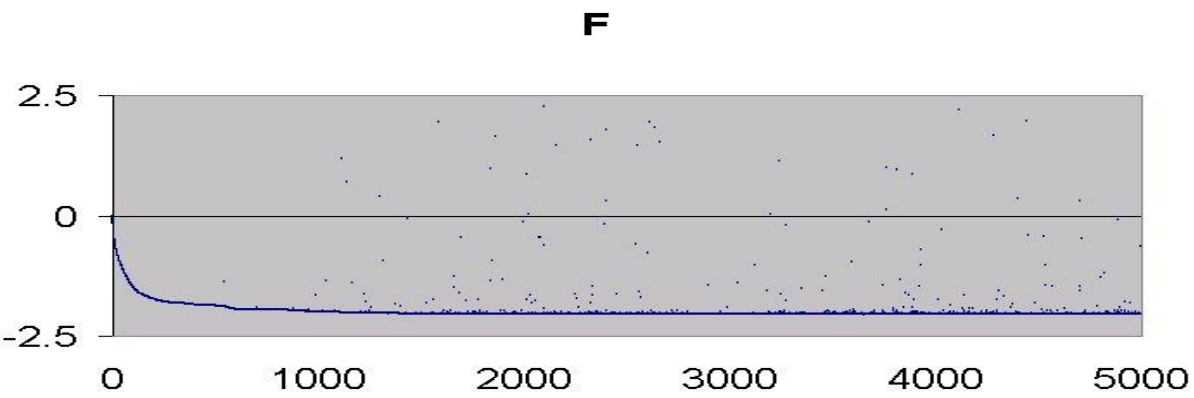


Funzionali di costo

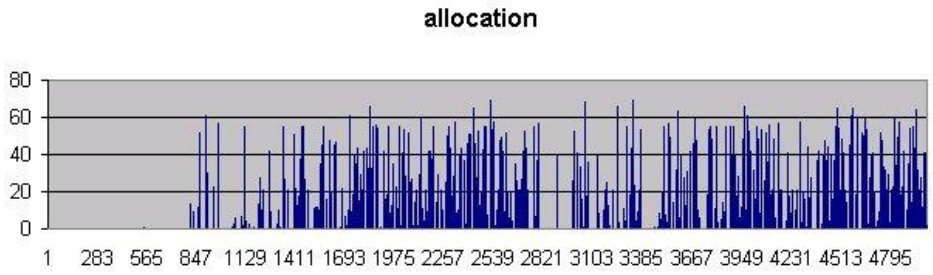
Funzionale	*Anticipa scadenze*	*Riduce schedula*	*Risolve violazioni*	*Uniforma carico*
BiasDeadline	SI	SI	NO	NO
Delay	SI per i task in ritardo	SI se task in ritardo	NO	NO
C_{max}	SI ma solo dell'ultimo	SI	NO	NO
Allocation	NO	NO	SI	NO
VarTotale	NO	NO	NO	SI

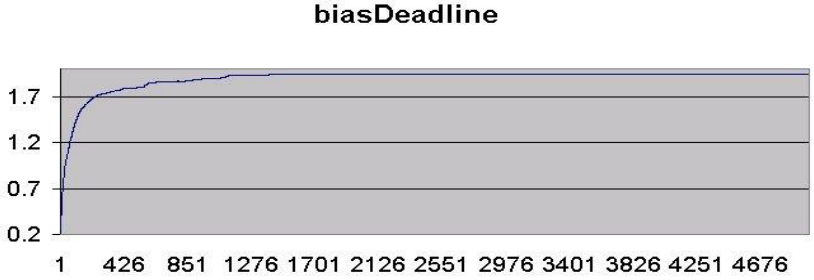
Andamento funzionali

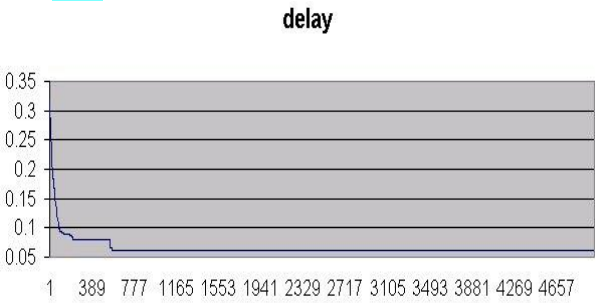
$F = K_A * allocation - K_B * \Delta biasDeadline + K_D * \Delta delay + K_V * \Delta varTotale + K_c * \Delta C_{max}$

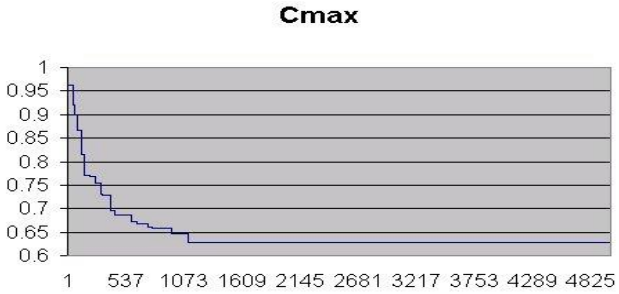


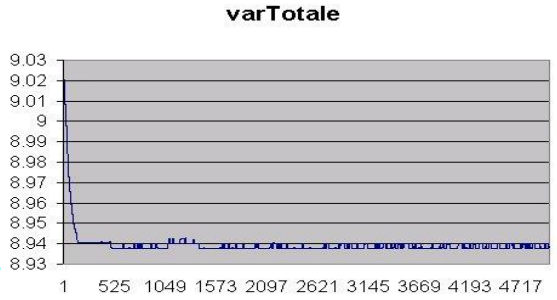
$$K_A = 10^{-3}$$
$$K_B = 10^{-7}$$
$$K_D = 10^{-7}$$
$$K_V = 10^{-11}$$
$$K_C = 10^{5}$$

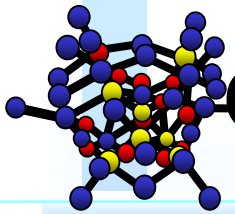






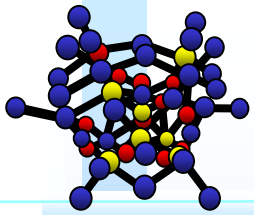






Comparison: Micro GRID for media IEEE Multimedia 2012

	AXMEDIS	MMGRID	MediaGrid	GridCast	MediaGrid.org	Omneon MediaGrid
Content Management: storage, UGC, ..	X		(x)	(x)	X	X
Content computing/processing: adaptation, processing conversion, cross media content packaging, ..	X		(x)	(x)	(x)	X
Content Delivery Network Management	X	X	X	X	X	
Metadata enrichment and reasoning	X					
Content Protection Management (CAS/DRM)	X					
Content Indexing and Querying, knowledge base	X			X		X
Semantic Computing Reasoning on user profiling, content descriptors, recommendations	X					
User Interaction Support, rendering, collaboration	X	X			X	
Client player as grid nodes for intelligent content	X					
Global and/or Local grid	L/(G)	G	G	G	G/L	L



sommario

- Contesto tecnologico
- Overhead e Speed-UP
- Aspetti Topologici per il calcolo parallelo
- Soluzioni GRID
- Soluzioni MicroGRID
- Apache Hadoop MapReduce
  - Vedere anche esercitazioni
  - http://www.disit.org/7073
- Apache Hadoop MapReduce 2
- Apache Spark

HDFS

Storage

MapReduce

Processing

The Apache™ Hadoop® project develops open-source Software for reliable, scalable, distributed computing

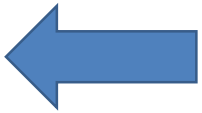
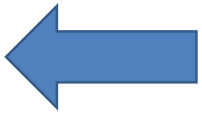
- Hadoop is an Open Source Platform designed and realized for processing, analyzing and storing big amounts of data, distributed and non-structured data.

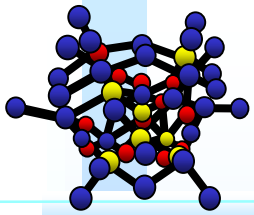
- Hadoop has been designed to be highly scalable: from single-host to thousands of hosts (*nodes*), with a high degree of *fault tolerance*.

- With Hadoop it is possible to analyze and process big datasets through highly scalable, distributed batch processing.

- Hadoop has been implemented in the first 2000 by Google, alla base della sua architettura architettura distribuita per l'indicizzazione dele Web.
- Currently, Hadoop is distributed as an Open Source project by the Apache Software Foundation → thousands of contributes from the community to improve its ***core technology***
- Key idea: instead of processing a big data block with a single node, Hadoop "fragments" Big Data in different blocks, so that every block can be processed in a parallel way by the different nodes composing the distributed architecture (***cluster***).

Hadoop Main Use Cases

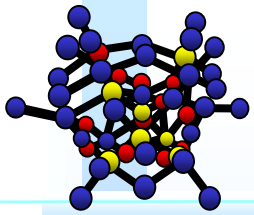
- Data-intensive text processing
- Assembly of large genomes
- Graph mining
- Machine learning and data mining 
- Large scale social network analysis 



Architetture MapReduce

- At the basis of Apache Hadoop solution
- Designed to realize
  - Large scale **distributed batch processing** infrastructures
  - Exploit low costs hardware from 1 to multiple cores, with low to large Mem, with low to large storage each
  - Covering huge (big data) storage that could not be covered by multi core parallel architectures
- See Yahoo! Hadoop tutorial
- https://developer.yahoo.com/hadoop/tutorial/index.html

Who is using Hadoop?

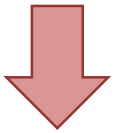


Typical problems of clusters

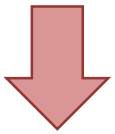
- Large number of nodes in the clusters
  - "You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done." – ***Leslie Lamport***
- Relevant probability of failures,
  - **CPU**: hundreds of thousands of nodes
  - **Network**: congestion may lead to do not provide data results and data inputs in times
  - **Network**: failure of apparatus
  - **Mem**: run out of space
  - **Storage**: run out of space, failure of a node, data corruption, failure of transmission
  - **Clock**: lack of synchronization, locked files and records not released, atomic transactions may lose connections and consistency
- Specific parallel solutions provide support for recovering from some of these failures

Distributed Systems: Data Storage

- Typically, specific resources are needed and provided for data storage (*Data Nodes*) as well as for the computational part (*Compute Nodes*)



During the execution of a process, data are copied in the Compute Nodes



This is OK for not so large amounts of data

How many data ?

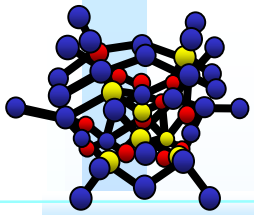
- Facebook
  - 500 TB each day
- Yahoo
  - More than 170 PB
- eBay
  - More than 6 PB

- The bottleneck is in moving/copying data to Computational Nodes

Hadoop peculiarity

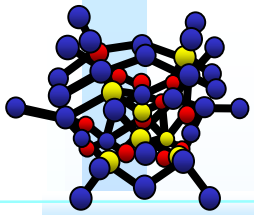
"Moving Computation is Cheaper than Moving Data"

It is not data to be moved/copied. Instead, the computational part is moved to the pertaining data portions to be processed



Typical problems

- **Hadoop has no security model**, nor safeguards against maliciously inserted data.
  - ♣ It cannot detect a man-in-the-middle attack between nodes
  - ♣ it is designed to handle very robustly
    - ➔ hardware failure
    - ➔ data congestion issues
- Storage may be locally become full:
  - ♣ Reroute, redistribute mechanisms are needed
  - ♣ Synchronization among different nodes is needed
  - ♣ This may cause:
    - ➔ network saturation
    - ➔ Deadlock in data exchange

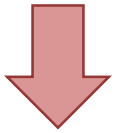


Recovering from failure

- If some node fails in providing results in time, the other nodes should do the work of the missing node
- The recovering process should be performed automatically
- The Solution may be not simple to implement in non simple parallel architectures, topologies, data distribution, etc.
- Limits:
  - Individual hard drives can only sustain read speeds between 60-100 MB/second
  - assuming four independent I/O channels are available to the machine, that provides 400 MB of data every second

Processes vs Data

- Different processes operate on specific portions of data. The single process allocation depends on the position of such data portions to be accessed and processed.



Data locality

  - Minimize data flow through the cluster nodes
  - Avoid communications among nodes to serve computational nodes
  - Hadoop is grounded on moving computation to the data And **not** data to computation !!!

Hadoop Scalability

- A significative performance increase may be perceived only with a high number of nodes.
- Parallel computational paradigms, such as MPI (Message Passing Interface) may have better performance on a cluster composed by 2, 4, or up to 10 thousand nodes.
- Hadoop is specifically designed to have a linear scalability curve.
- If a process is running on (for example) 10 nodes, it can work efficiently as well on thousands of nodes without causing network congestion.

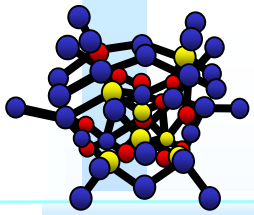
UNIVERSITÀ DEGLI STUDI FIRENZE

DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

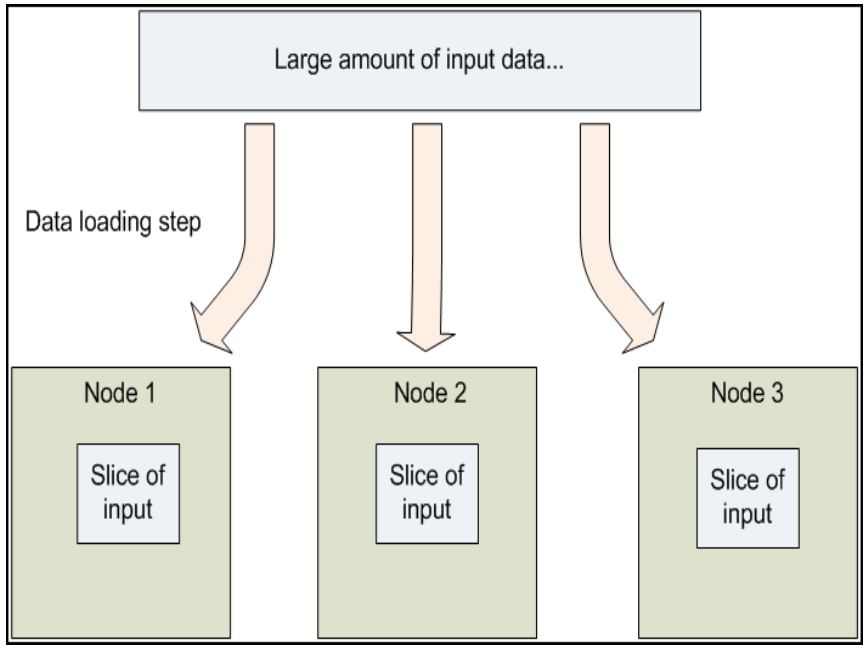
Hadoop Architecture

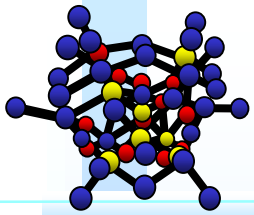
- The distributed Hadoop architecture is a *master-slave* architecture, composed by two main components:
  - **HDFS** Hadoop Distributed File System (HDFS) for data storage.
  - **MapReduce** as a parallel computational paradigm.



Hadoop data distribution

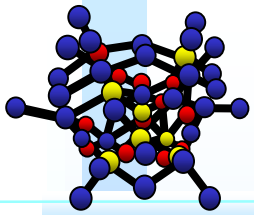
- Is based on the Hadoop Distributed File System (HDFS) that distribute data files on large chunks on the different nodes
- Each chunk is replicated across several nodes, thus supporting the failure of nodes.
  - An active process maintains the replications when failures occurs and when new data are stored.
  - Replicas are not instantly maintained aligned !!!





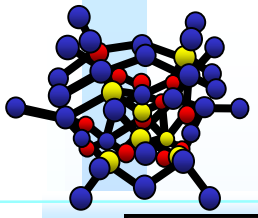
Processes vs Data

- Processes works on specific chunks of data. The allocations of processes depend on the position of the chunks they need to access,
- That is: data locality.
  - This reduces the data flow among nodes
  - Avoid communications among nodes to serve computational nodes
  - Hadoop is grounded on moving computation to the data !!!
    And **not** data to computation.

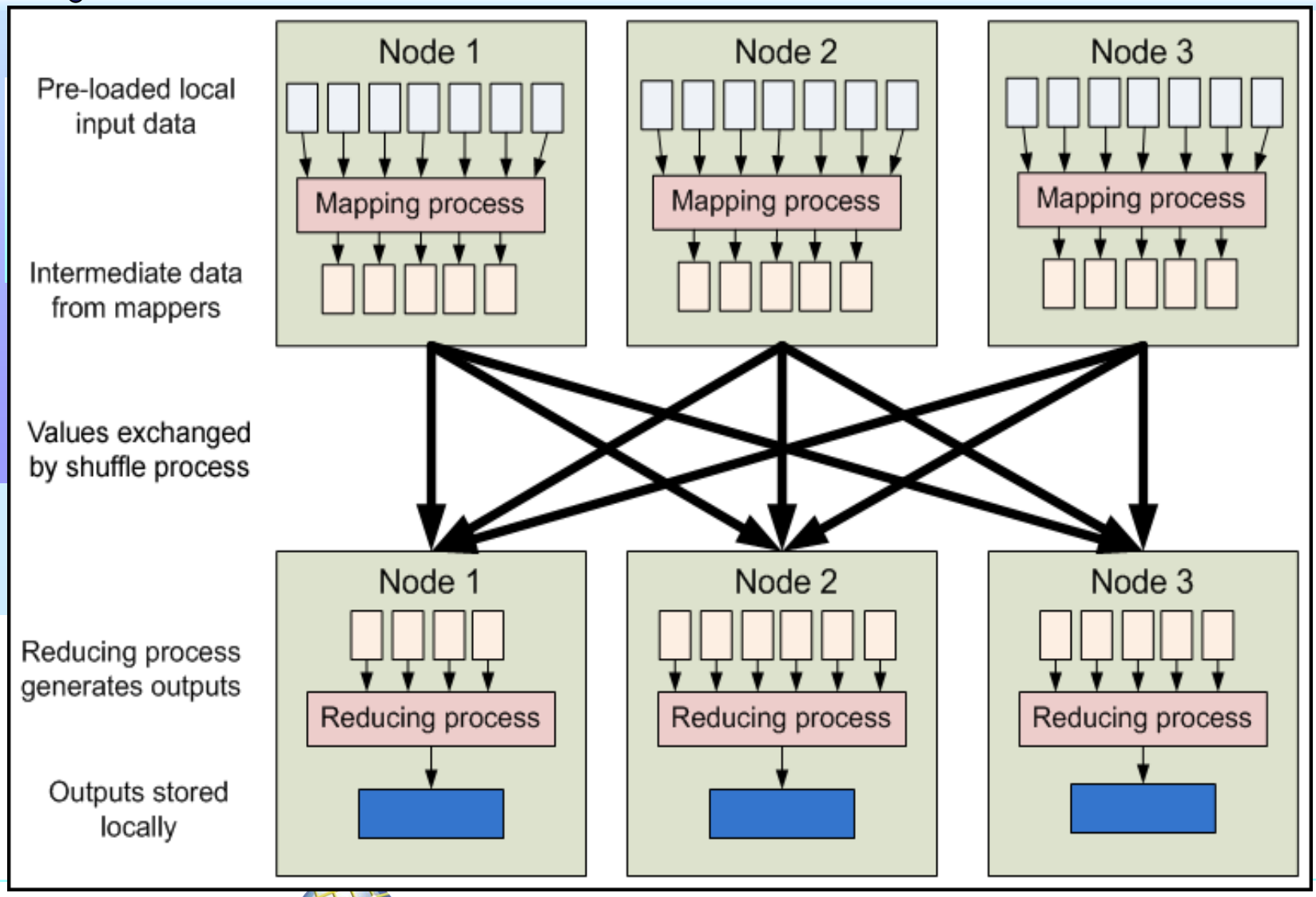


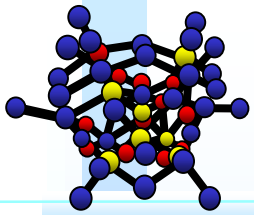
MapReduce: Isolated Processes

- The main idea:
  - Each individual record is processed by a task in isolation from one another. Replications allow to reduce communications for: contour conditions, data overlap, etc.
- This approach does not allow any program to be executed.
- Algorithms have to be converted into parallel implementations to be executed on cluster of nodes.
- This programming model is called MapReduce model



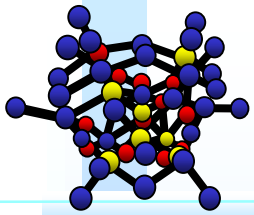
Mappers and Reducers





How it works

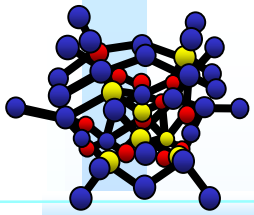
- Programmer has to write the Mappers and the Reducers
- The data migration
  - from Nodes hosting mappers' outputs
  - to nodes needing those data to processing Reducers
  - is automatically implicitly performed if these data is specifically tagged
- Hadoop internally manages all of the data transfer and cluster topology issues.
- This is quite different from traditional parallel and GRID computing where the communications have to be coded may be with MPI, RMI, etc.



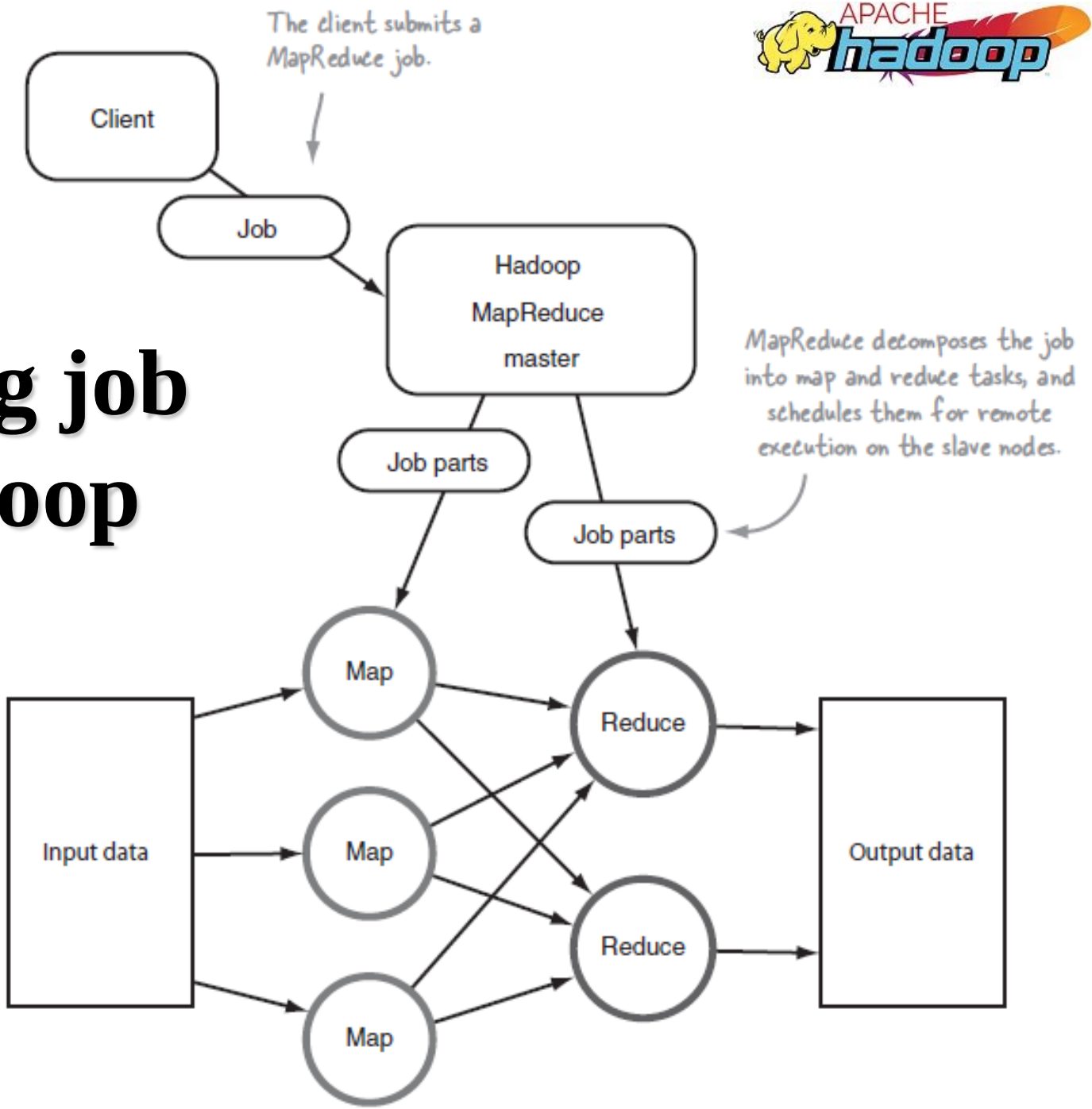
Hadoop Flat Scalability

- Hadoop on a limited amount of data on a small number of nodes may not demonstrate particularly stellar performance as the overhead involved in starting Hadoop programs is relatively high.
- parallel/distributed programming paradigms such as MPI (Message Passing Interface) may perform much better on two, four, or perhaps a dozen machines.

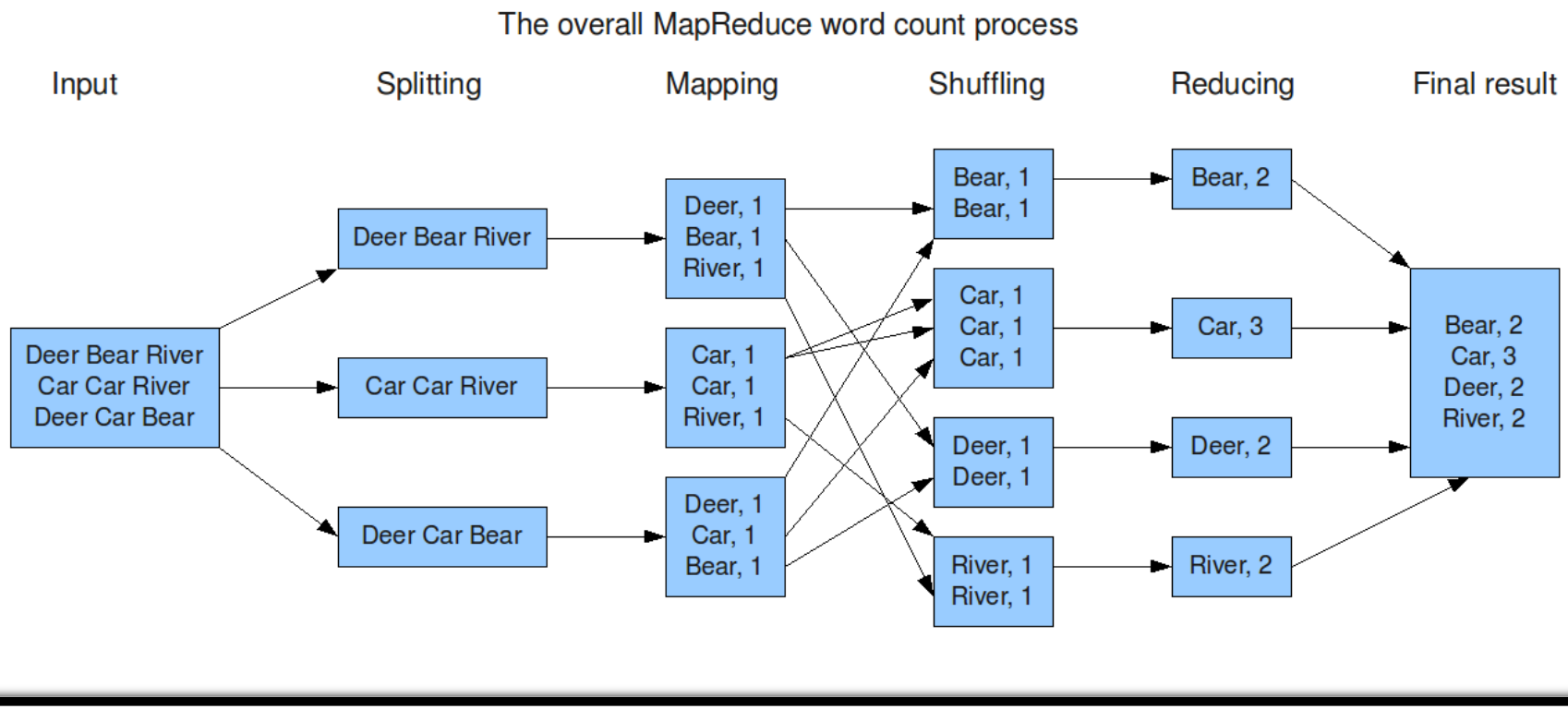
- specifically designed to have a very flat scalability curve
- If it is written for 10 nodes may work on thousands with small rework effort

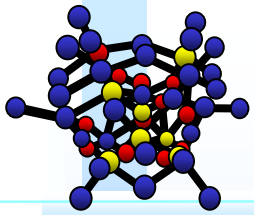


Releasing job on Hadoop



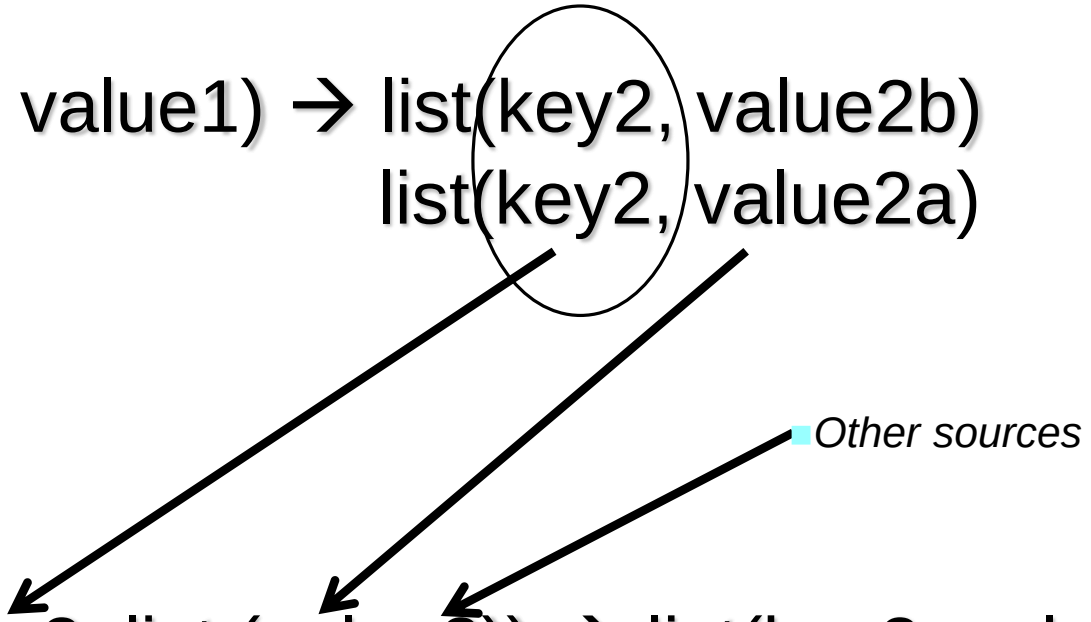
MapReduce: Word count



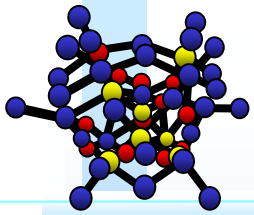


An example of WordCount

- Mapper
  - map(key1, value1) → list(key2, value2b)
    list(key2, value2a)



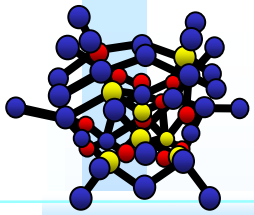
- Reducer:
  - reduce (key2, list (value2)) → list(key3, value3)
- When the mapping phase has completed, the intermediate (key, value) pairs must be exchanged between machines to send all values with the same key to a single reducer.



Mapping input and ouput

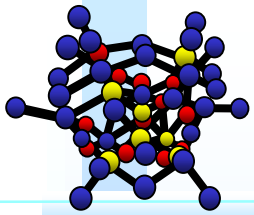
- Document1:
  - Queste sono le slide di Mario Rossi
- Document2:
  - Le slide del corso di Sistemi Distribuiti

- Mapping output: → list(key2, value2)
  - Queste, Document1
  - sono, Document1
  - le, Document1
  - slide, Document1
  - di, Document1
  - Mario, Document1
  - Rossi, Document1
  - Le, Document2
  - slide, Document2
  - del, Document2
  - corso, Document2
  - di, Document2
  - Sistemi, Document2
  - Distribuiti, Document2



Reduce input

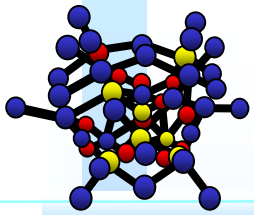
- reduce (key2, list (value2)) → **list(key3, value3)**
  - Queste, Document1
  - sono, Document1
  - le, Document1
  - slide, list (Document1, Document2)
  - di, list (Document1, Document2)
  - Mario, Document1
  - Rossi, Document1
  - Le, Document2
  - corso, Document2
  - Sistemi, Document2
  - Distribuiti, Document2



Reducer Output as ……

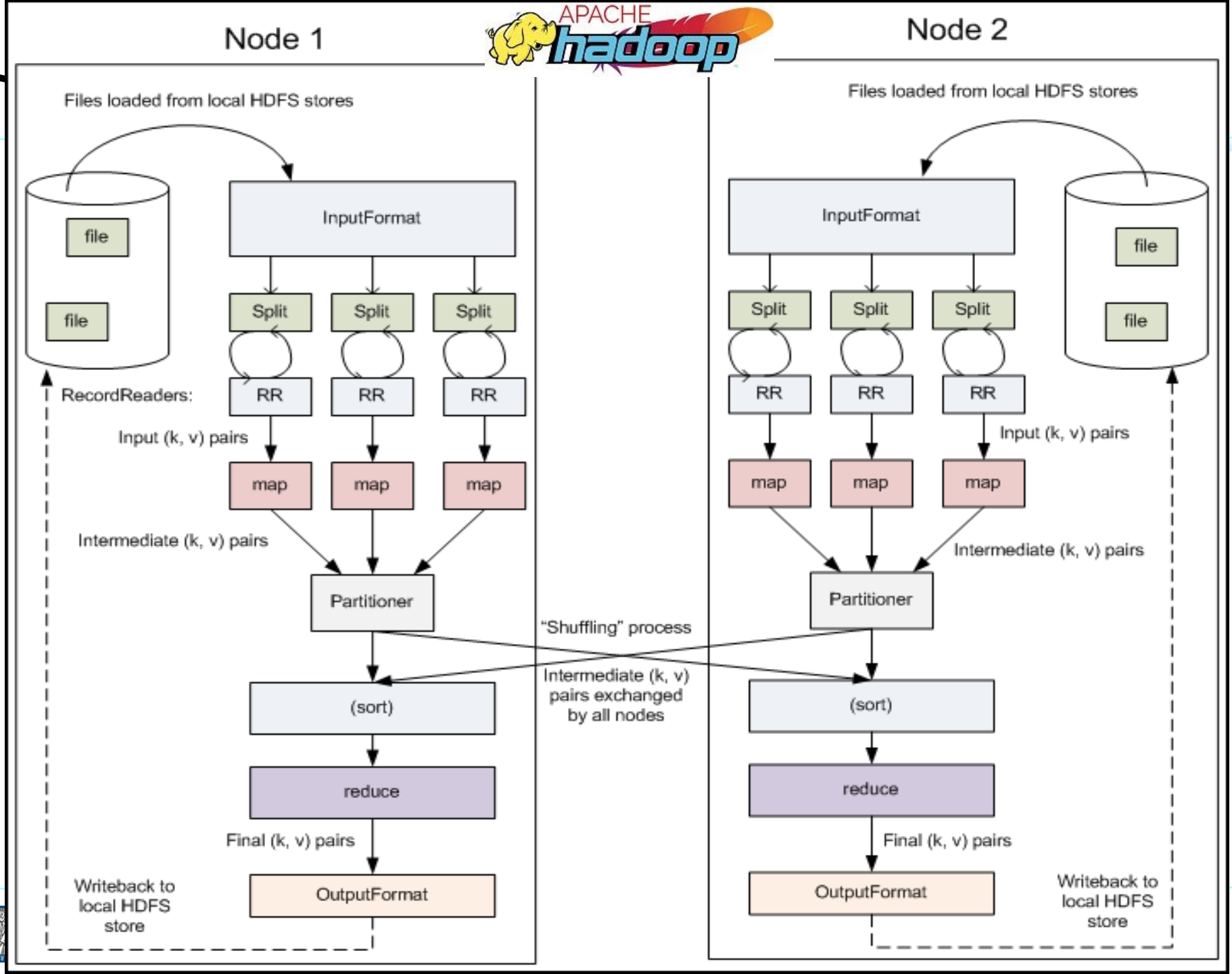
- inverted index (the previous example)
  - …
  - le, Document1
  - slide, list (Document1, Document2)
  - di, list (Document1, Document2)
  - …
- Counting Words
  - …
  - le, 1
  - slide, 2
  - di, 2
  - …

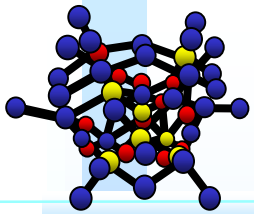
```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)

reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```



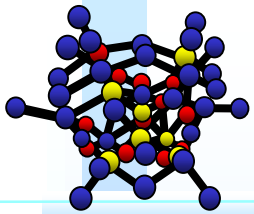
pertanto

- Gli esempi sono due
- 1) Conteggio delle parole nei vari testi, distribuzione della frequenza delle parole
  - Tipicamente usato in algoritmi di stima della similarita' in base al numero di parole simili ed al loro peso (frequenza in un certo dominio, documento)
- 2) generazione di una lista di sorgenti per ogni parala.
  - Esempio di indice inverso

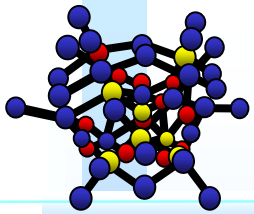

Node 1
Node 2
APACHE hadoop
Files loaded from local HDFS stores
Files loaded from local HDFS stores
file
file
file
file
InputFormat
InputFormat
Split
Split
Split
Split
Split
Split
RecordReaders:
RR
RR
RR
RR
RR
RR
Input (k, v) pairs
Input (k, v) pairs
map
map
map
map
map
map
Intermediate (k, v) pairs
Intermediate (k, v) pairs
Partitioner
Partitioner
"Shuffling" process
Intermediate (k, v) pairs exchanged by all nodes
(sort)
(sort)
reduce
reduce
Final (k, v) pairs
Final (k, v) pairs
OutputFormat
OutputFormat
Writeback to local HDFS store
Writeback to local HDFS store



```
public static class MapClass extends MapReduceBase
    implements Mapper <LongWritable, Text, Text, IntWritable> {
     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();
     public void map (LongWritable key, Text value,
                      OutputCollector<Text, IntWritable> output,
                      Reporter reporter) throws IOException {
      String line = value.toString(); // da text to line string
      StringTokenizer itr = new StringTokenizer(line); // tokenized
      while (itr.hasMoreTokens()) { // finche vi sono token nella lista
        word.set(itr.nextToken()); // identifica il token e mettilo in WORD
        output.collect(word, one); // assegna 1 al WORD
      }
    }
```



```
/*** A reducer class that just emits the sum of the input values. */
 public static class Reduce extends MapReduceBase
   implements Reducer <Text, IntWritable, Text, IntWritable> {

   public void reduce (Text key, Iterator<IntWritable> values,
                       OutputCollector<Text, IntWritable> output,
                       Reporter reporter) throws IOException {
     int sum = 0;
     while (values.hasNext()) { // finche vi sono elementi in Values
       sum += values.next().get(); // fai la somma
     }
     output.collect(key, new IntWritable(sum));
   }
 }
```



driver

```
public void run(String inputPath, String outputPath) throws Exception
{
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
     // the keys are words (strings)
    conf.setOutputKeyClass(Text.class);
    // the values are counts (ints)
    conf.setOutputValueClass(IntWritable.class);
     conf.setMapperClass(MapClass.class);
    conf.setReducerClass(Reduce.class);
     FileInputFormat.addInputPath(conf, new Path(inputPath));
    FileOutputFormat.setOutputPath(conf, new Path(outputPath));
     JobClient.runJob(conf);
 }
```

When you extend the MapReduce mapper class you specify the key/value types for your inputs and outputs. You use the MapReduce default InputFormat for your job, which supplies keys as byte offsets into the input file, and values as each line in the file. Your map emits Text key/value pairs.

```
public static class Map
    extends Mapper<LongWritable, Text, Text, Text> {
```

A Text object to store the document ID (filename) for your input

```
  private Text documentId;
```

To cut down on object creation you create a single Text object, which you'll reuse.

```
  private Text word = new Text();
```

Extract the filename from the context

```
  @Override
  protected void setup(Context context) {
    String filename =
        ((FileSplit) context.getInputSplit()).getPath().getName();
    documentId = new Text(filename);
  }
```

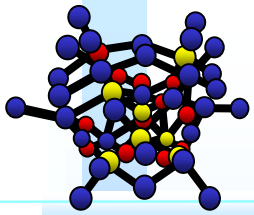
This method is called once at the start of the map and prior to the map method being called. You'll use this opportunity to store the input filename for this map.

This map method is called once per input line; map tasks are run in parallel over subsets of the input files.

```
  @Override
  protected void map(LongWritable key, Text value,
                     Context context)
      throws IOException, InterruptedException {
    for (String token :
            StringUtils.split(value.toString())) {
      word.set(token);
      context.write(word, documentId);
    }
  }
}
```

For each word your map outputs the word as the key and the document ID as the value.

Your value contains an entire line from your file. You tokenize the line using StringUtils (which is far faster than using String.split).



Reducer

Much like your Map class you need to specify both the input and output key/value classes when you define your reducer.

```
public static class Reduce
    extends Reducer<Text, Text, Text, Text> {
  private Text docIds = new Text();
  public void reduce(Text key, Iterable<Text> values,
                     Context context)
      throws IOException, InterruptedException {

    HashSet<Text> uniqueDocIds = new HashSet<Text>();

    for (Text docId : values) {

      uniqueDocIds.add(new Text(docId));
    }

    docIds.set(new Text(StringUtils.join(uniqueDocIds, ",")));

    context.write(key, docIds);
  }
}
```

The reduce method is called once per unique map output key. The Iterable allows you to iterate over all the values that were emitted for the given key.

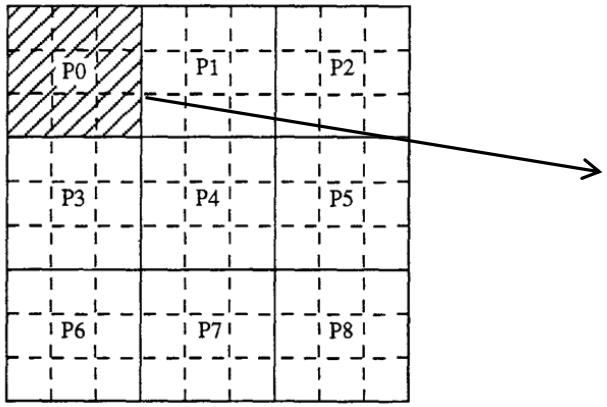
Keep a set of all the document IDs that you encounter for the key.

Iterate over all the DocumentIDs for the key.

Add the document ID to your set. The reason you create a new Text object is that MapReduce reuses the Text object when iterating over the values, which means you want to create a new copy.

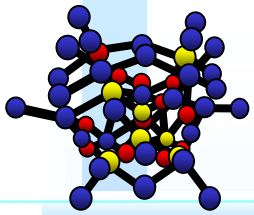
Your reduce outputs the word, and a CSV-separated list of document IDs that contained the word.

Esempio di elaborazione locale spaziale



1	2	3
4	5	6
7	8	9

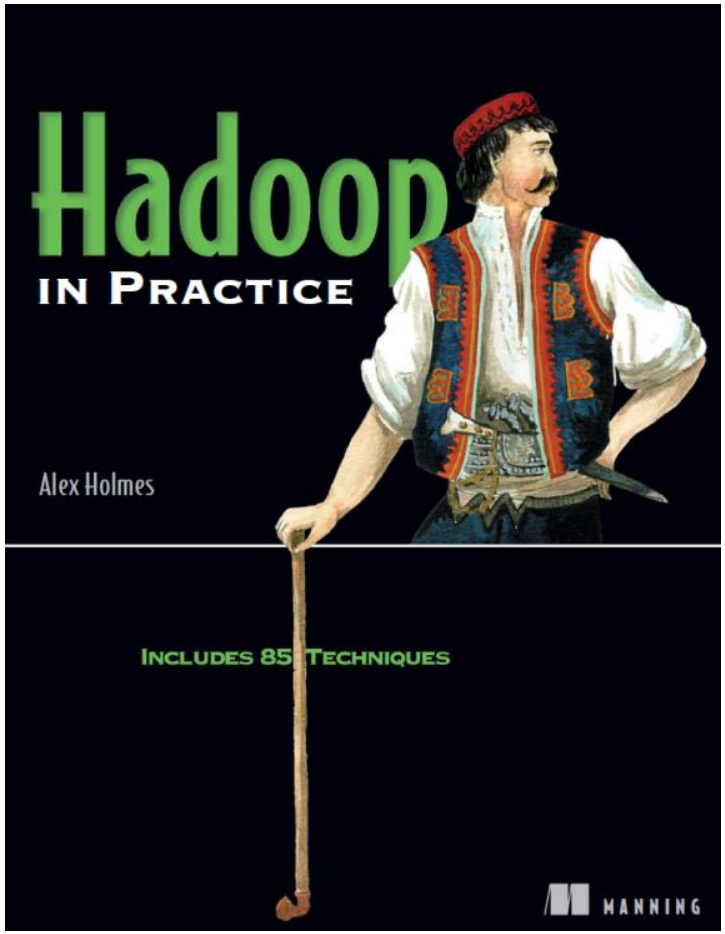
- *Map (arrange data and put lables/keys):*
  - *Extract submatrix from file*
  - *Create 3x3 (or NxN) data with a single Key*
- *Reduce:*
  - *Ex A) Make the estimation of average*
  - *Ex B) make filtering*
  - *Ex c) perform derivative for optical flow, etc.*
  - *Etc. etc.*

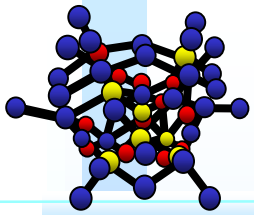


Reference

- Yahoo tutorial
  - https://developer.yahoo.com/hadoop/tutorial/index.html

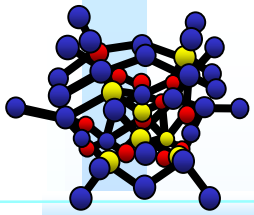
- Book:
  - ALEX HOLMES, *Hadoop in Practice*, 2012 by Manning Publications Co. All rights reserved.





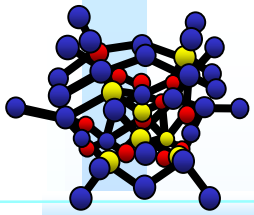
sommario

- Contesto tecnologico
- Overhead e Speed-UP
- Aspetti Topologici per il calcolo parallelo
- Soluzioni GRID
- Soluzioni MicroGRID
- Apache Hadoop MapReduce
  - Vedere anche esercitazioni
  - http://www.disit.org/7073
- Apache Hadoop MapReduce 2
- Apache Spark

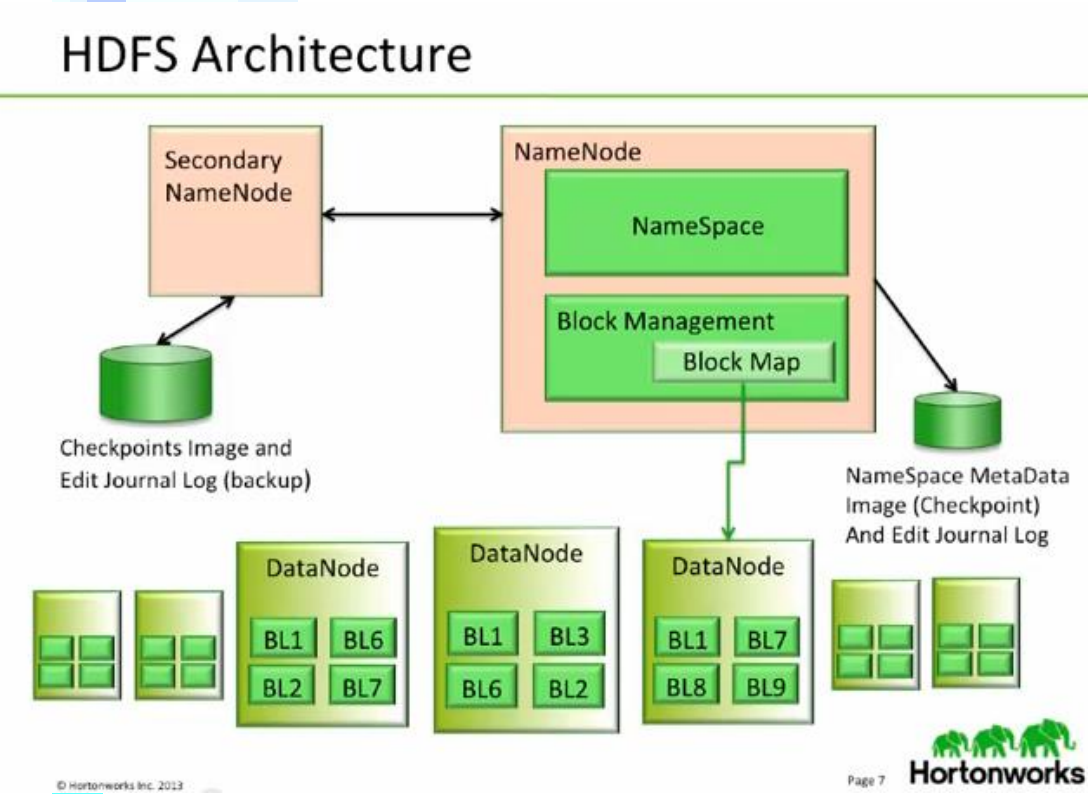


From Hadoop 1 to 2

- Hadoop 1
  - Based on HDFS and Map Reduce
    - a single Namenode managed the entire namespace for a Hadoop cluster
  - Suitable for
    - Large scale distributed processing, SIMD
    - Data on the Storage HDFS
  - UnSuitable for:
    - Iterative processing
    - Memory intensive computing
    - Executing multitenacy processes



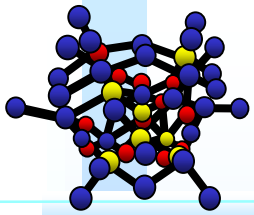
HDFS: Hadoop File System



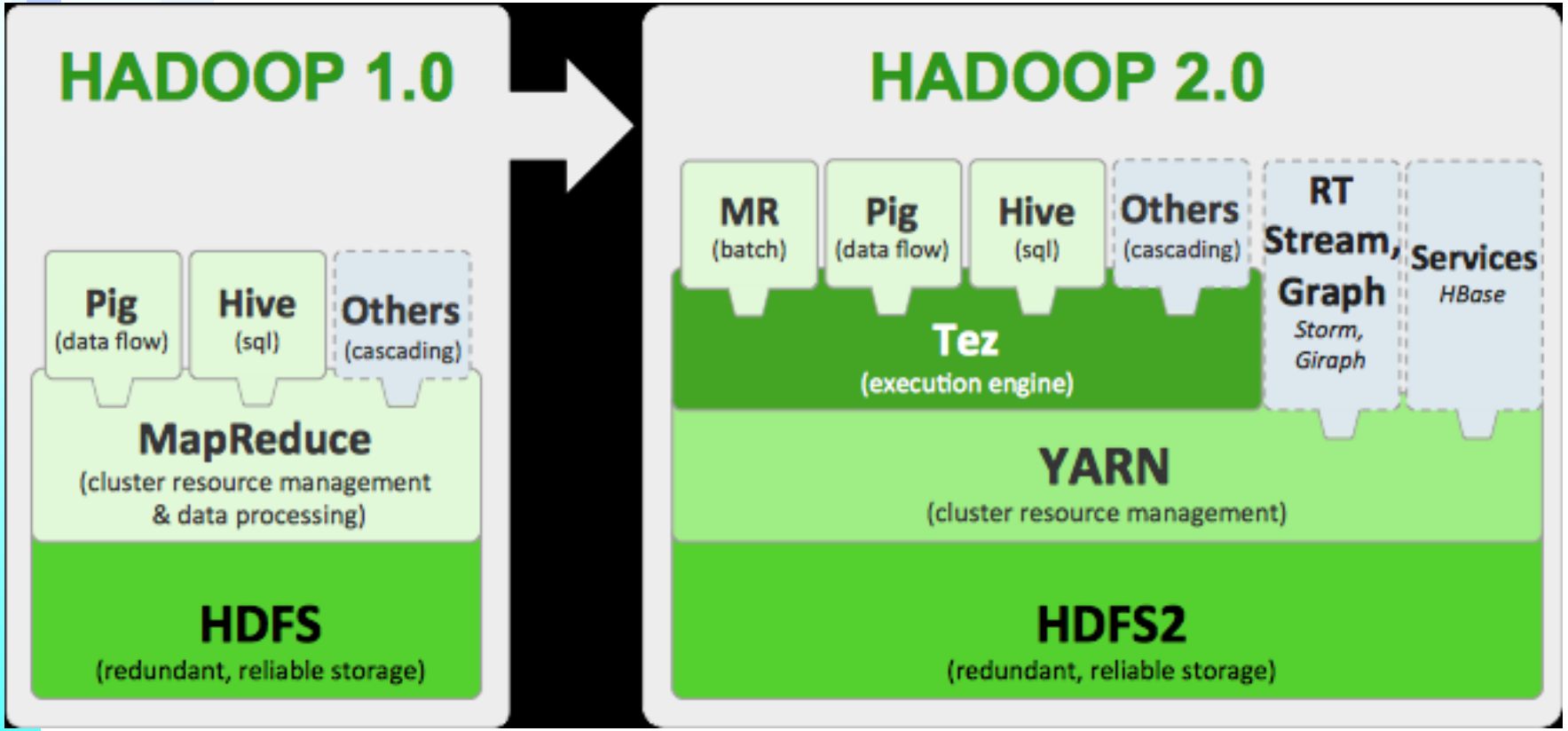
- Namespaces and blocks storage service.
  - namespace service: manages operations on files and directories, such as creating and modifying files and directories.
  - block storage service: implements data node cluster management, block operations and replications

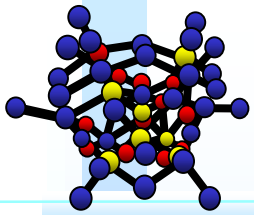
In HD1:

- a single Namenode manages the entire namespace for a Hadoop cluster



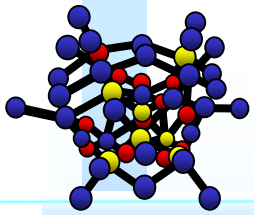
From Hadoop 1.0 to Hadoop 2.0



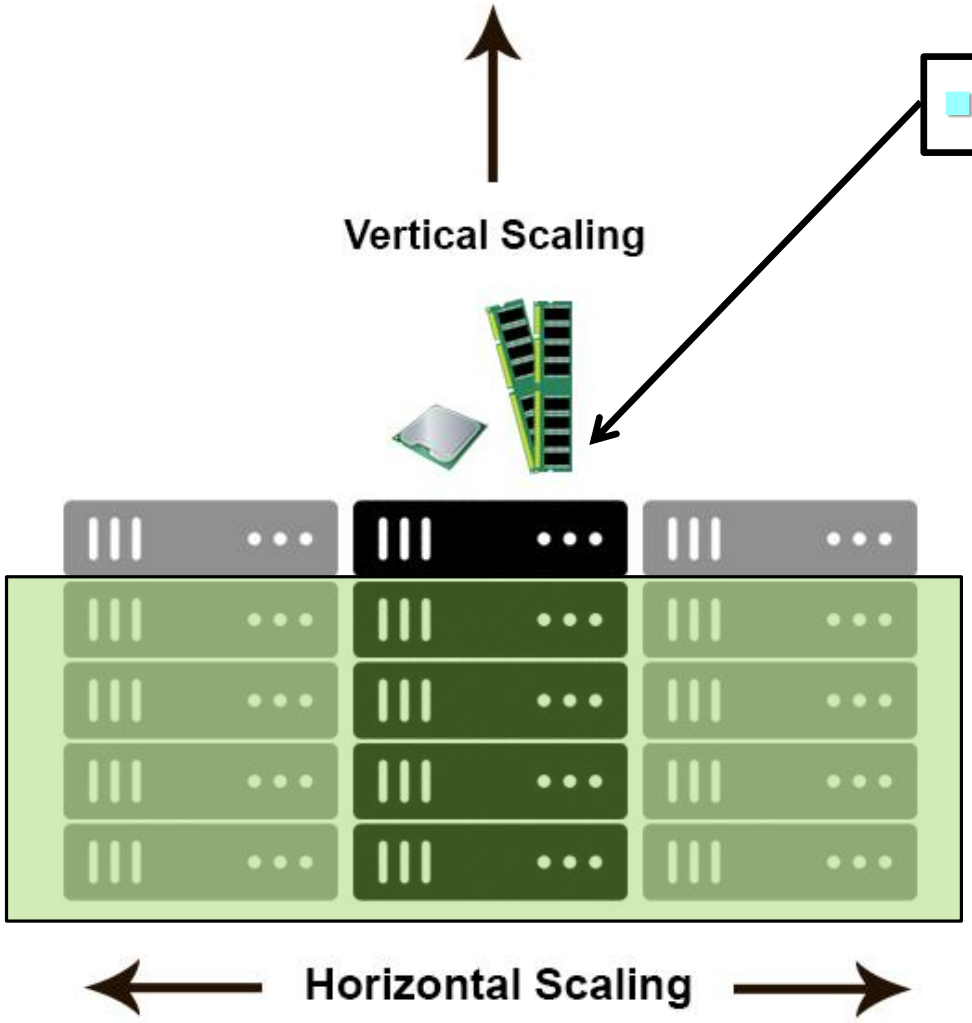


HDFS: Hadoop 2 File System

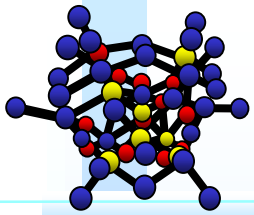
- HDFS federation,
  - multiple Namenode servers manage namespaces
    - **horizontal scaling**, performance improvements, and multiple namespaces.
    - existing Namenode configurations to run without changes.
- Moving from HDFS1 to HDFS federation requires
  - formatting Namenodes,
  - updating to use the latest Hadoop cluster software,
  - adding additional Namenodes to the cluster.



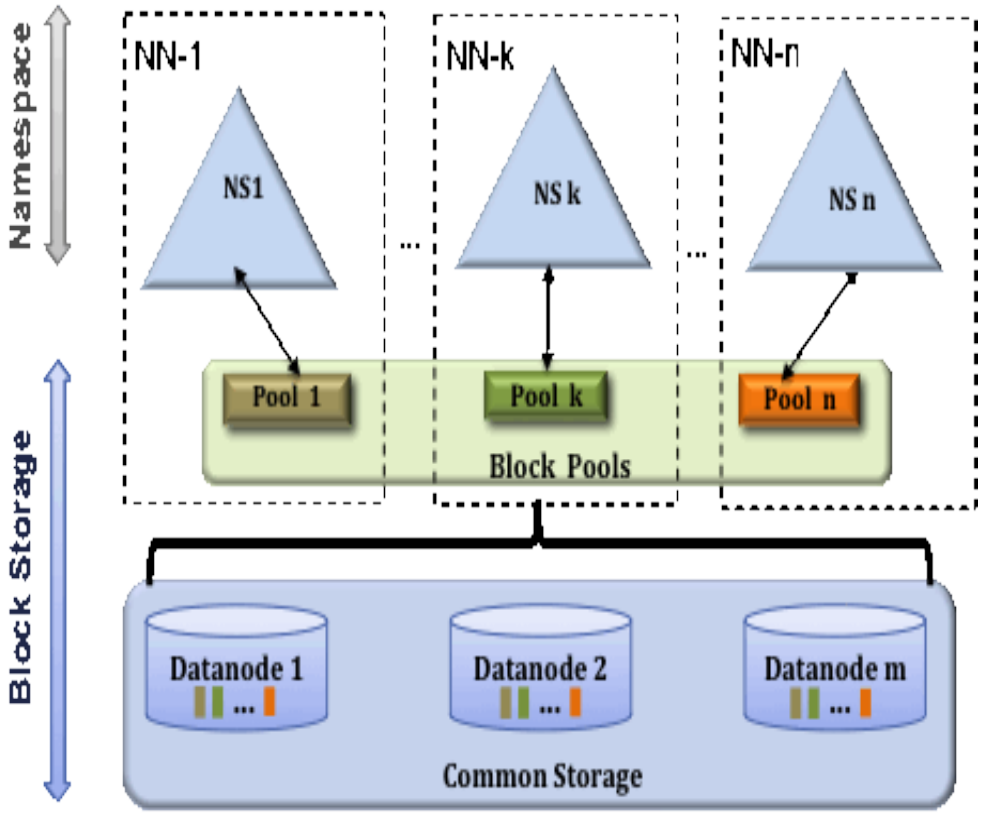
Horizontal vs Vertical Scalability



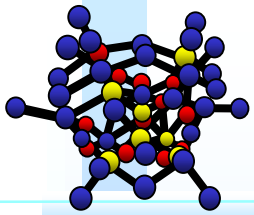
- **Vertical Scalability**
  - More memory
  - More clock per second
  - More CPU into the server (cluster)
  - More storage
- **Horizontal Scalability**
  - More Servers into your resource pool/area
  - (More Clusters)



Multiple Namenodes/Namespaces

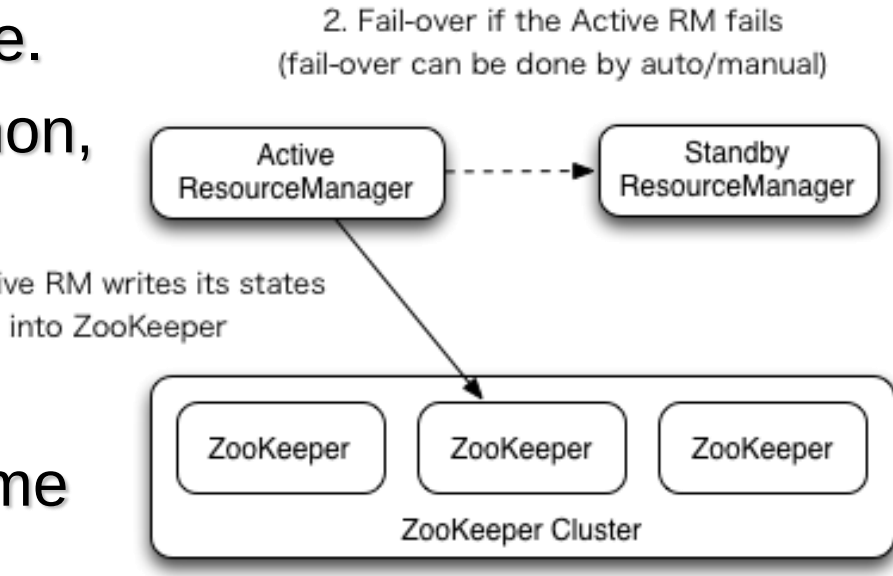


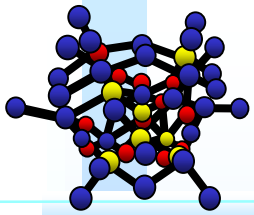
- federation uses multiple independent Namenodes/namespaces.
- **Namenodes** are federated, independent and do not require coordination with each other.
- **Datanodes** are used as common storage for blocks by all the Namenodes.
  - Each Datanode registers with all the Namenodes in the cluster.
  - Datanodes send periodic heartbeats and block reports, handle commands from the Namenodes.



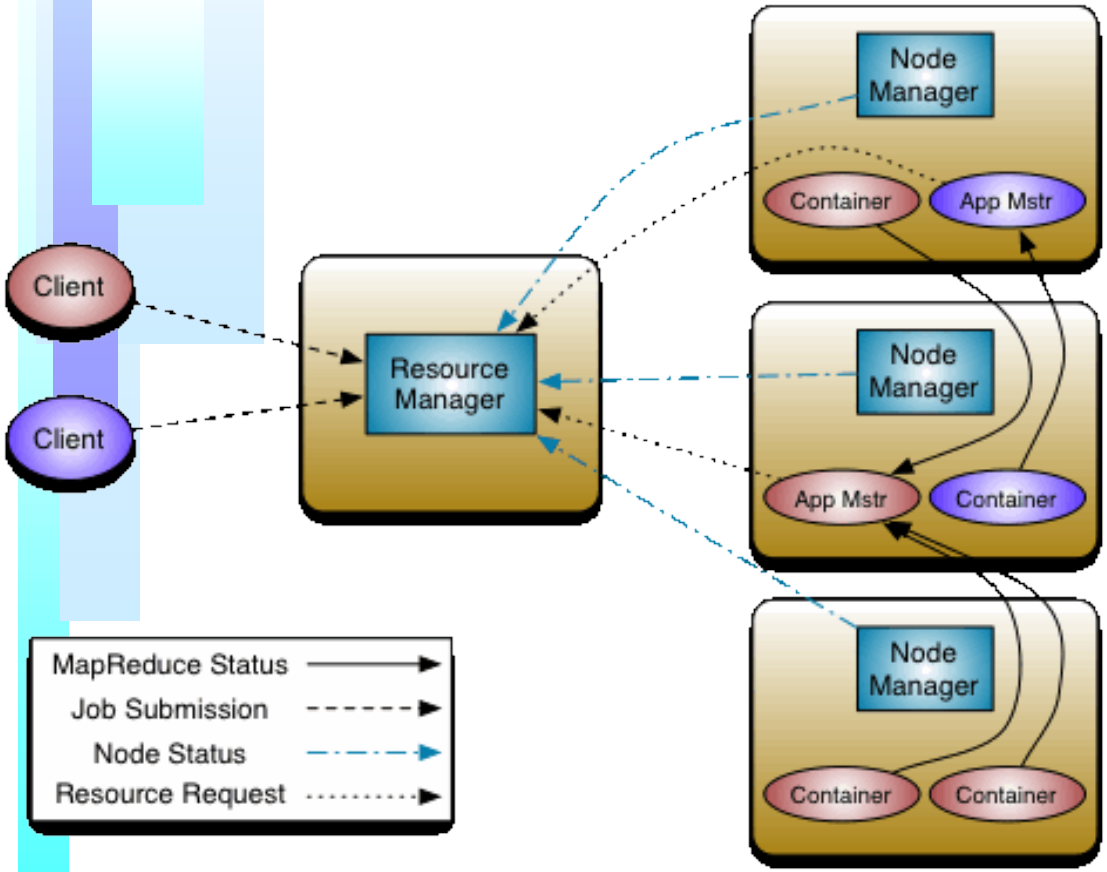
Introduction of YARN→ HD3

- **Yarn is a data operating system**
  - is a resource manager separating
    - →Job/Process scheduler/monitoring
    - →Resource Management (RM)
- **Yarn Introduces *true multi tenancy*.**
  - multiple jobs at the same time.
  - Multiple languages: java python, ruby, ..
  - security layer on the HDFS
  - high availability on Hadoop
  - Resource reservation over time

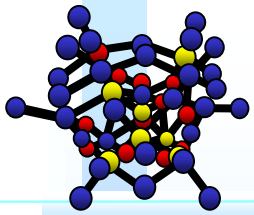




Apache Hadoop YARN

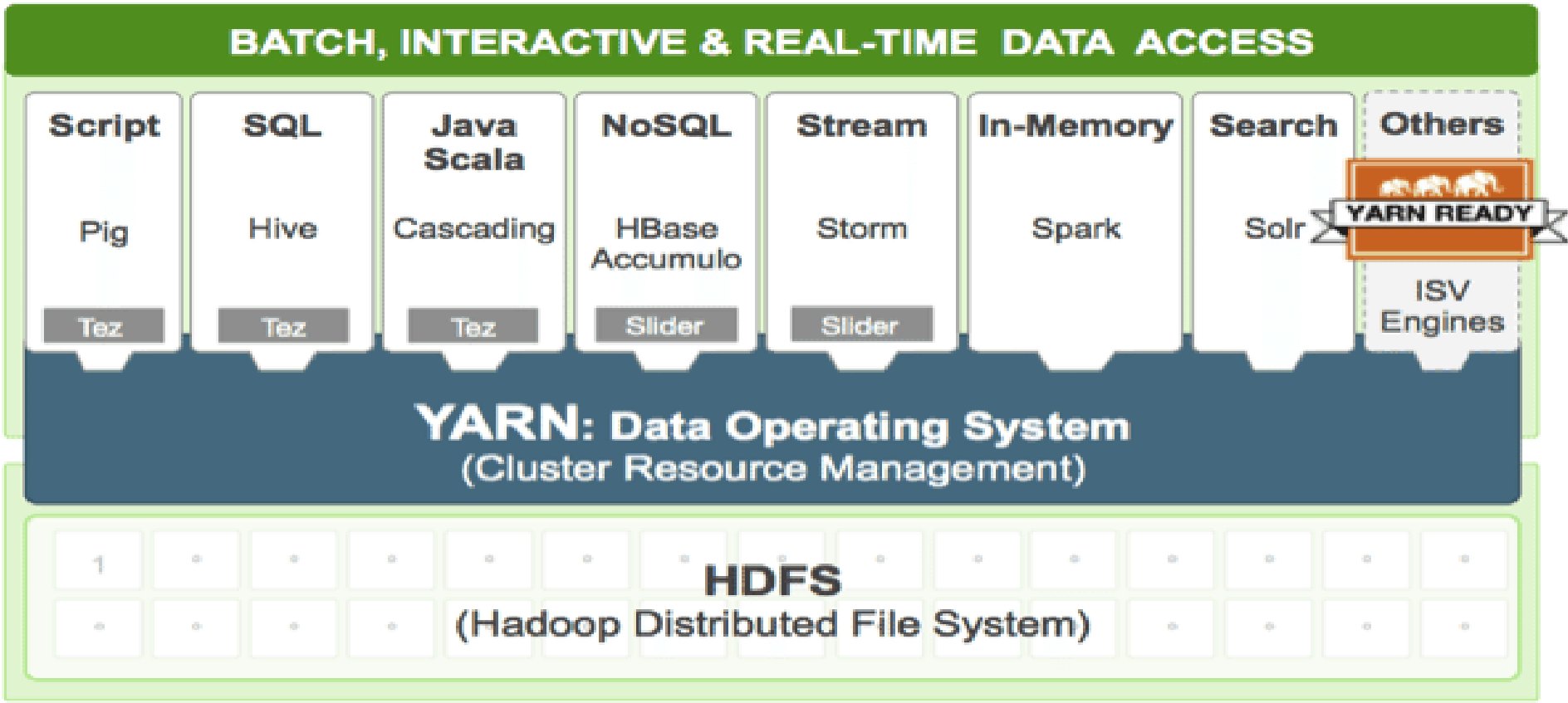


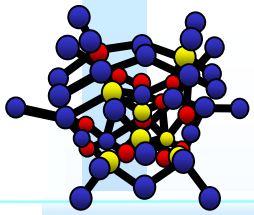
- The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the Resource Manager/Scheduler.



Yarn

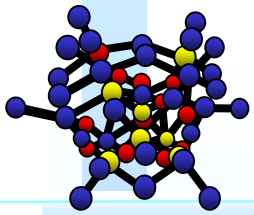

APACHE
hadoop


BATCH, INTERACTIVE & REAL-TIME DATA ACCESS
Script
Pig
Tez
SQL
Hive
Tez
Java
Scala
Cascading
Tez
NoSQL
HBase
Accumulo
Slider
Stream
Storm
Slider
In-Memory
Spark
Search
Solr
Others
YARN READY
ISV
Engines
YARN: Data Operating System
(Cluster Resource Management)
HDFS
(Hadoop Distributed File System)



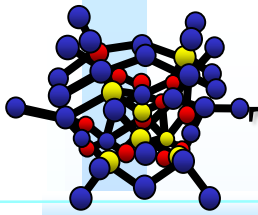
sommario

- Contesto tecnologico
- Overhead e Speed-UP
- Aspetti Topologici per il calcolo parallelo
- Soluzioni GRID
- Soluzioni MicroGRID
- Apache Hadoop MapReduce
  - Vedere anche esercitazioni
  - http://www.disit.org/7073
- Apache Hadoop MapReduce 2
- Apache Spark

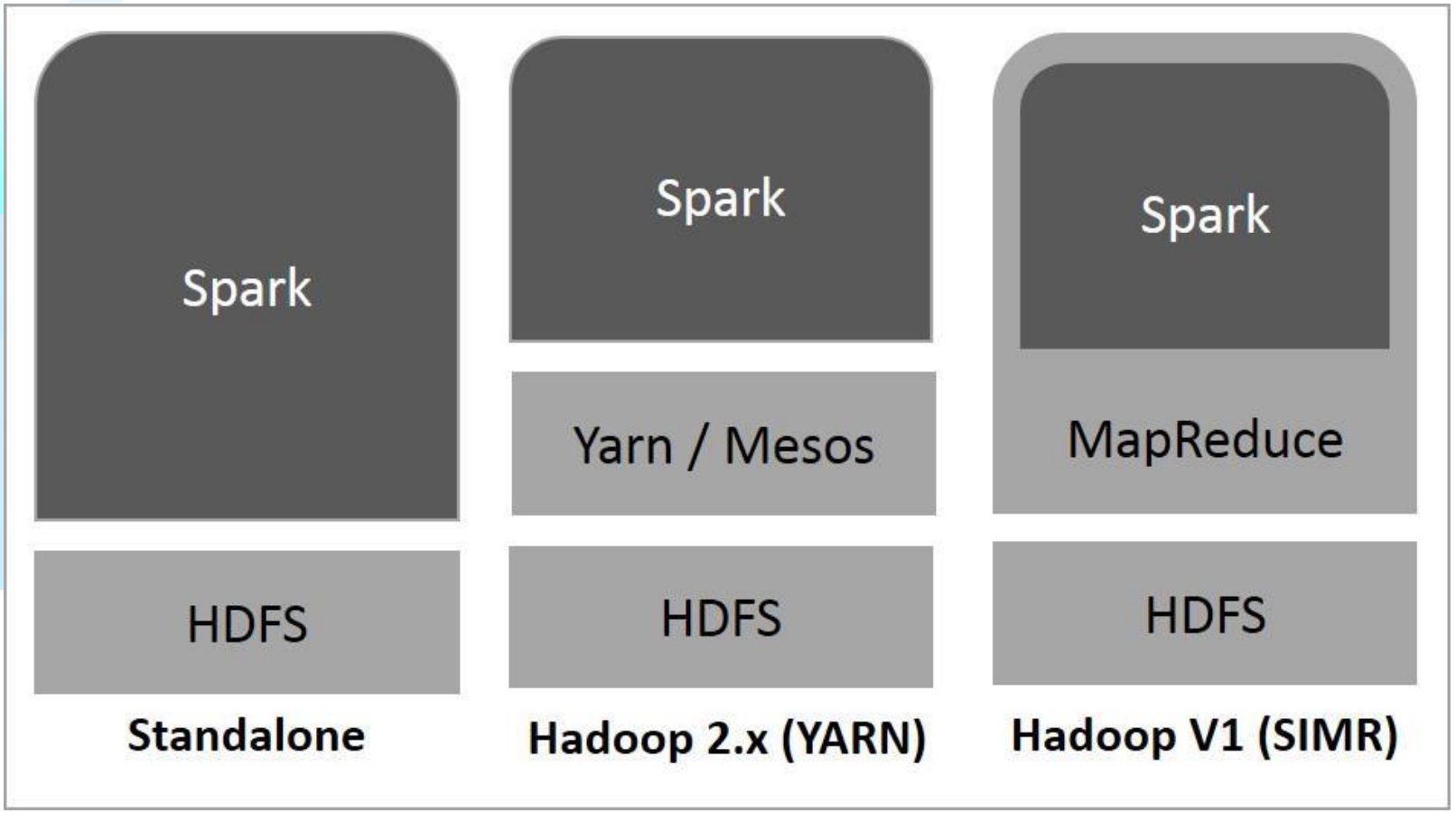


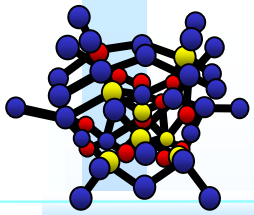
Apache Spark

- Most of the time on **Hadoop MapReduce algorithms** is spent on data read/write on disk, shuffle, etc.
  - Apache Spark is proposing a solution to work on processes that use data in memory → 100x faster
- **Lack of some «SQL»** of Hadoop, Hbase, HDFS
  - Apache Spark provides support for SQL
- **Write applications quickly** in:
  - Java, Scala, Python, R
- **Additional feature** Library/Support
  - MLIB: machine learning
  - GraphX: graphics library
  - Streaming: Streaming support, Spark Streaming
  - R: R Support: R-Spark

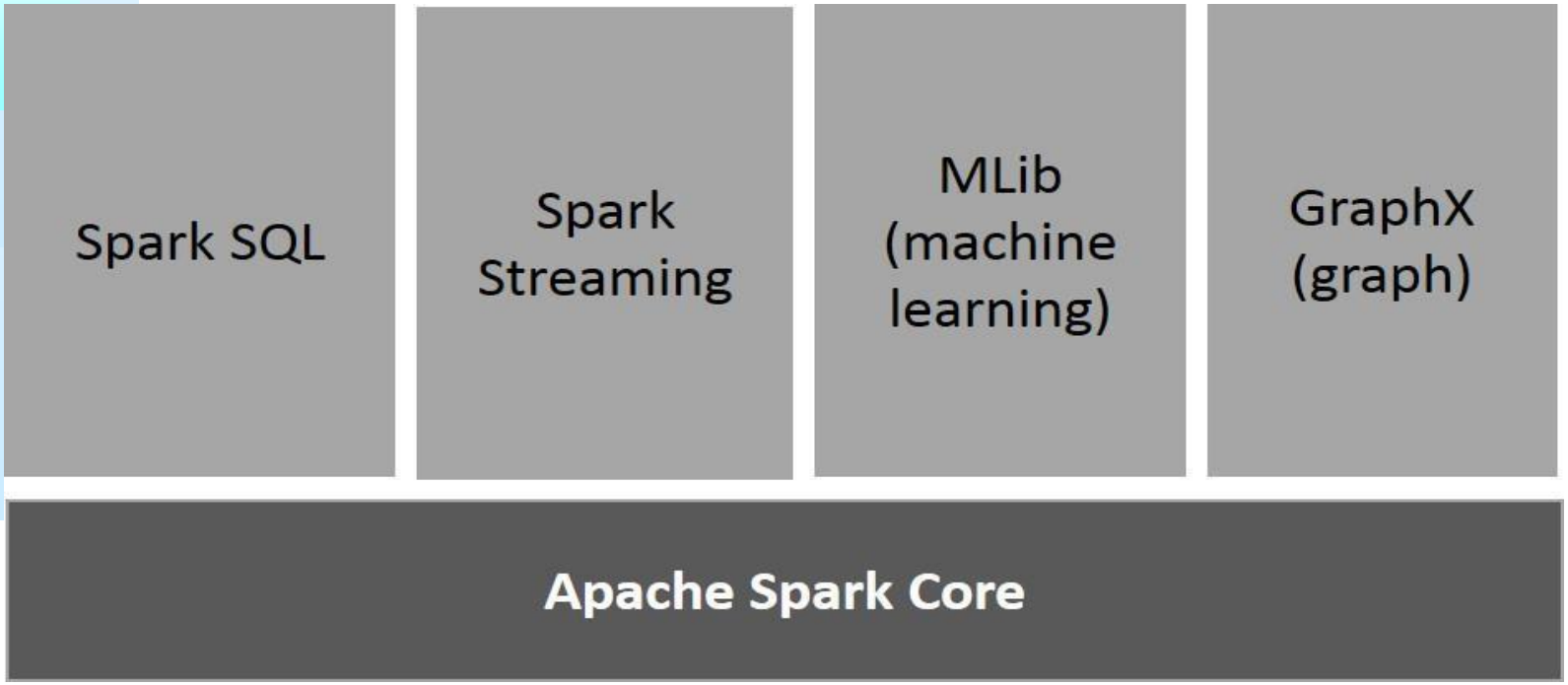


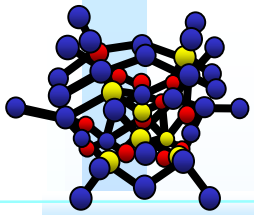
Three ways of Spark deployment





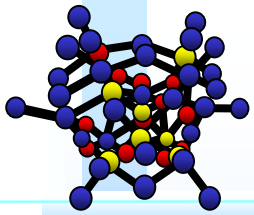
Main Spark Components



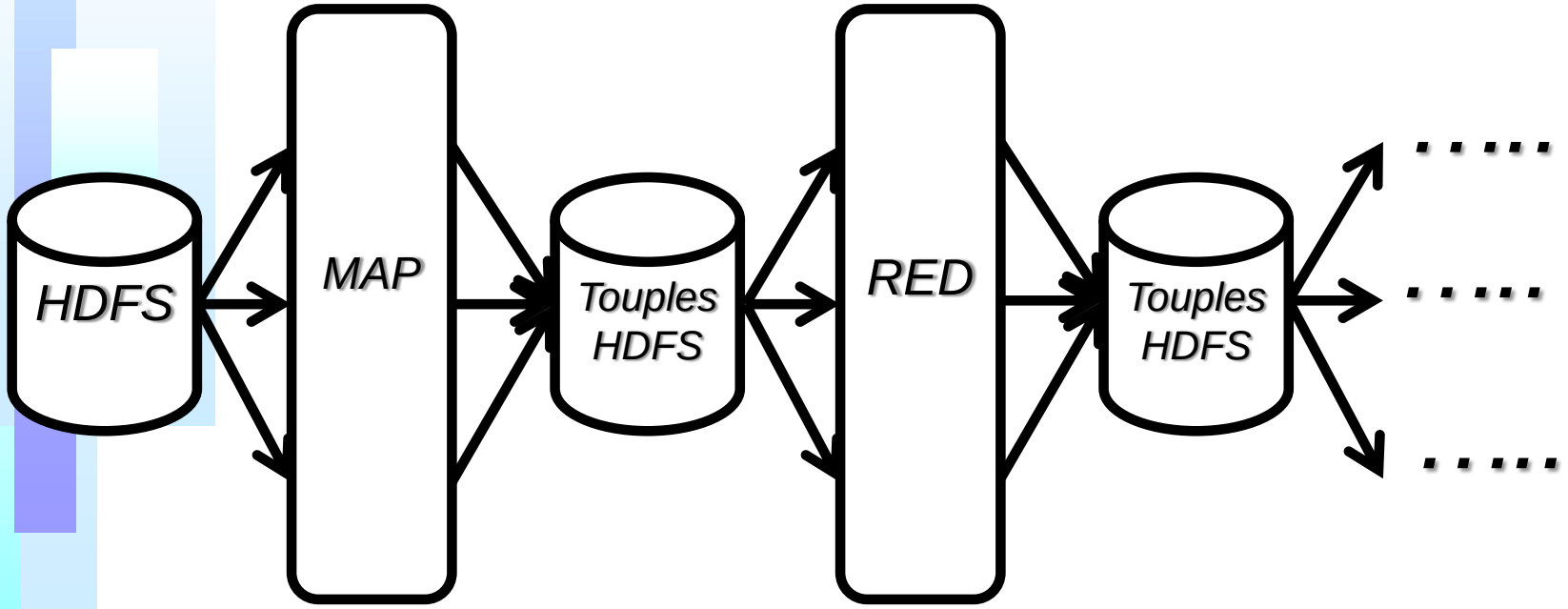


Spark Core

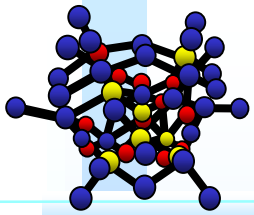
- Spark allows to perform **in-memory computing** and referencing datasets in external storage systems.
- **Spark SQL** introduce a new data abstraction called Resilient Distributed Datasets (RDD) on which is possible to perform some query access.
- **Spark Streaming** allows to perform analytics on the stream:
  - The analytics are written as small programs
- **Spark MLlib** a distributed (machine learning library) is on top of Spark
  - Mahout gained 10X with Spark from Hadoop
- **Spark GraphX** a distributed graph processing on top of Spark for Business Intelligence.



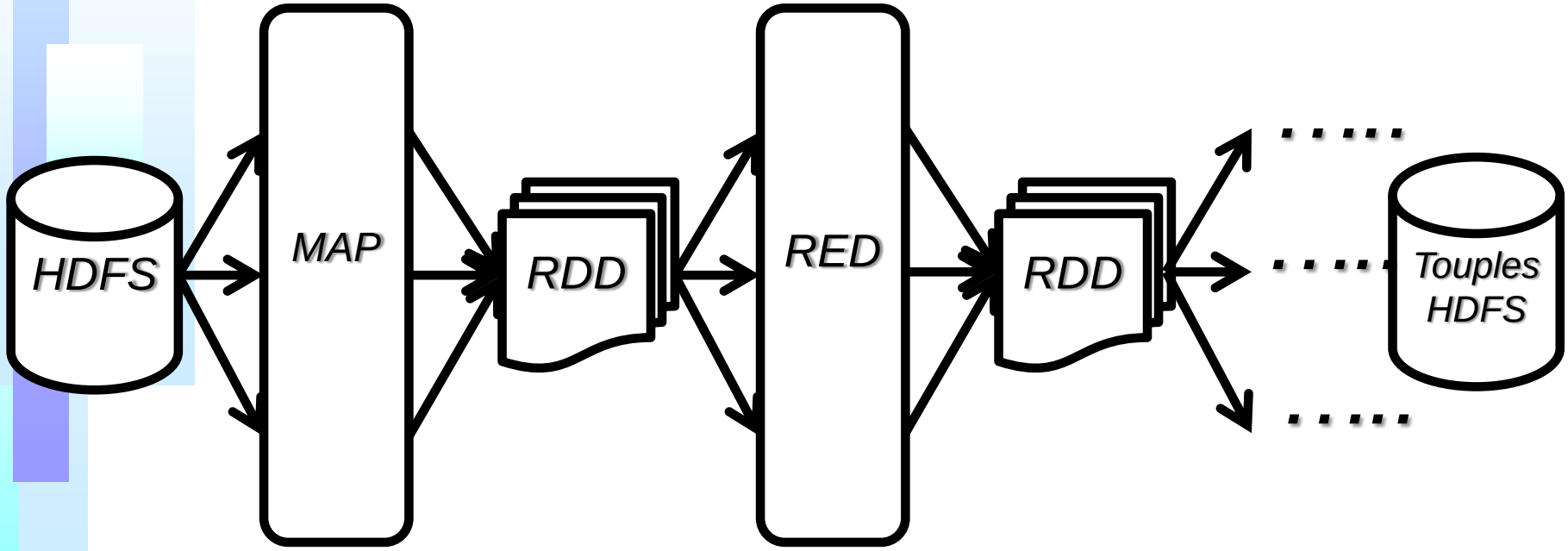
Classic MapReduce



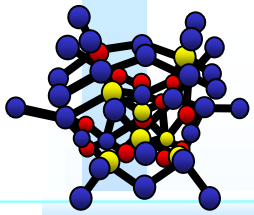
- In processes with several iterations the R/W cost may be too high even if performed in parallel



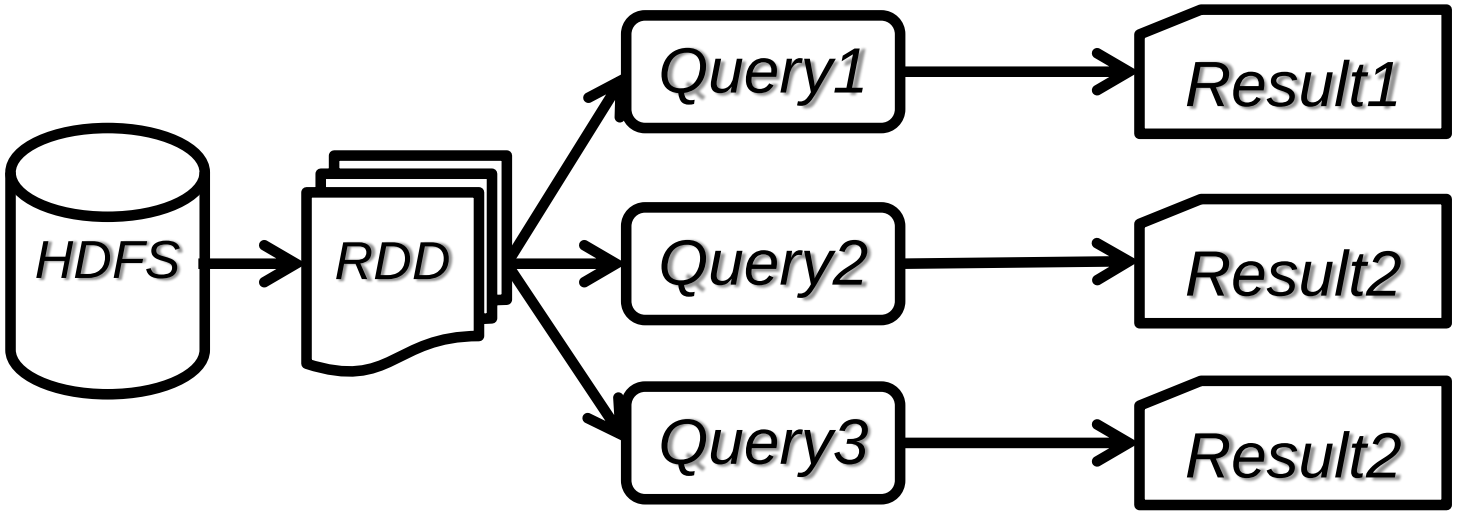
Iterations with Spark



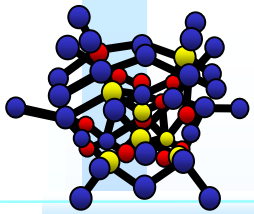
- In processes with several iterations the R/W is going to performed in memory and only finally on HDFS



Interactive Operation with Spark



- For Data Lake, Database Virtualization
- For Business analytics
- For dashboarding
- **Possible to persist** an RDD in memory



references

- *P. Bellini, I. Bruno, D. Cenni, P. Nesi, "Micro grids for scalable media computing and intelligence on distributed scenarious", IEEE Multimedia, Feb 2012, Vol.19, N.2, pp.69.79, IEEE Computer Soc. Press*
- *P. Bellini, M. Di Claudio, P. Nesi, N. Rauch, "Tassonomy and Review of Big Data Solutions Navigation", in "Big Data Computing", Ed. Rajendra Akerkar, Western Norway Research Institute, Norway, Chapman and Hall/CRC press, ISBN 978-1-46-657837-1, eBook: 978-1-46-657838-8, july 2013*
- *ALEX HOLMES, Hadoop in Practice, 2012 by Manning Publications Co. All rights reserved.*
- *I. Foster, C. Kesselman, The Grid, 2nd ed. Morgan Kaufmann, 2004.*
- *F. Berman, G. Fox, T. Hey, Grid Computing, Wiley, 2003.*
- *Burkhardt J. , et al., Pervasive Computing, Addison Wesley, 2002.*
- *Hansmann U., Merk L., Nicklous M.S., Stober T., Pervasive Computing, Springer Professional Computing, 2nd ed., 2003.*
- *A. S. Tanenbaum, M. Van Steen, "Distributed Systems", Prentice Hall, 2002*
- *http://www.globus.org*
- *http://www.axmedis.org*
- *http://www.cs.wisc.edu/condor*