29th October 2008

Systematic Cooperation in P2P Grids

Cyril Briquet

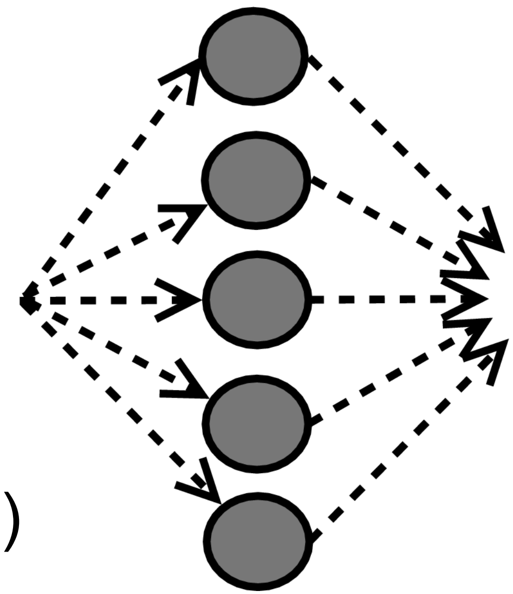
Doctoral Dissertation in Computing Science
Department of EE & CS (Montefiore Institute)
University of Liège, Belgium

Application class: Bags of Tasks

- Bag of Task = set of independent computational Tasks

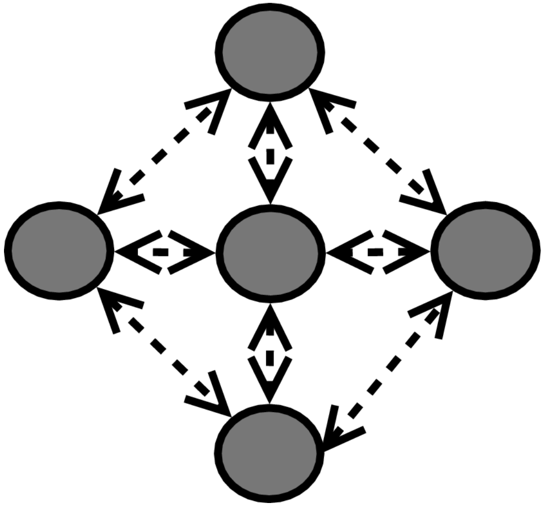
many domains:

- bioinformatics
- computer vision
- data mining
- distributed discrete-event simulation
- GIS, spatial indexing
- medical image processing (tomography)
- protein folding & docking
- search engine crawling & indexation



Application class: Iterative Stencil

- Iterative Stencil = inter-communicating computational Tasks, with iterative computations (sync. points)



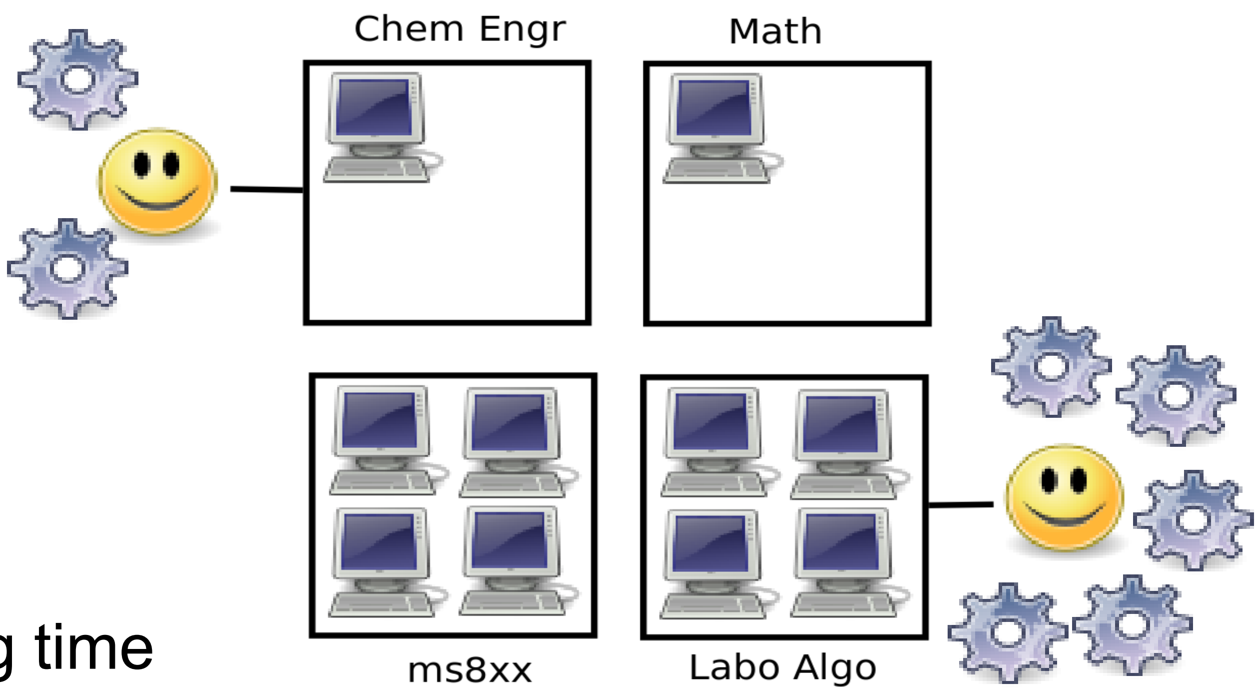
- system speed = slowest Task
  => load balancing required
- failure of any Task = restart everything, from the start => uninterrupted co-allocation required
- typical domains: CFD, electromagnetics

Human users + computational Tasks + no money for expensive infrastructure + limited number of desktop computers = ???

~~cluster computing~~

~~desktop computing~~

~~volunteer computing~~



Grid computing

- sharing of computing time
- separate organizations
- + fully decentralized and automated... => **P2P Grid computing**

P2P Grids operate in an environment too dynamic for most human users

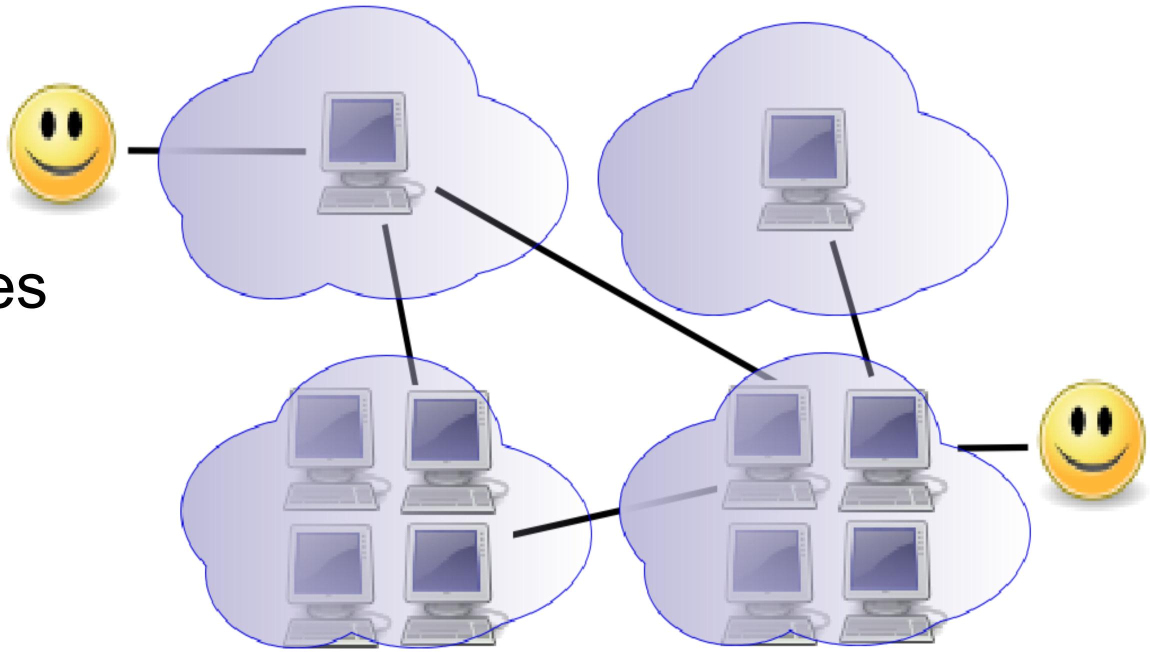
human users and administrators do expect
short response times and a simple interface

complexity of the P2P Grid should be hidden

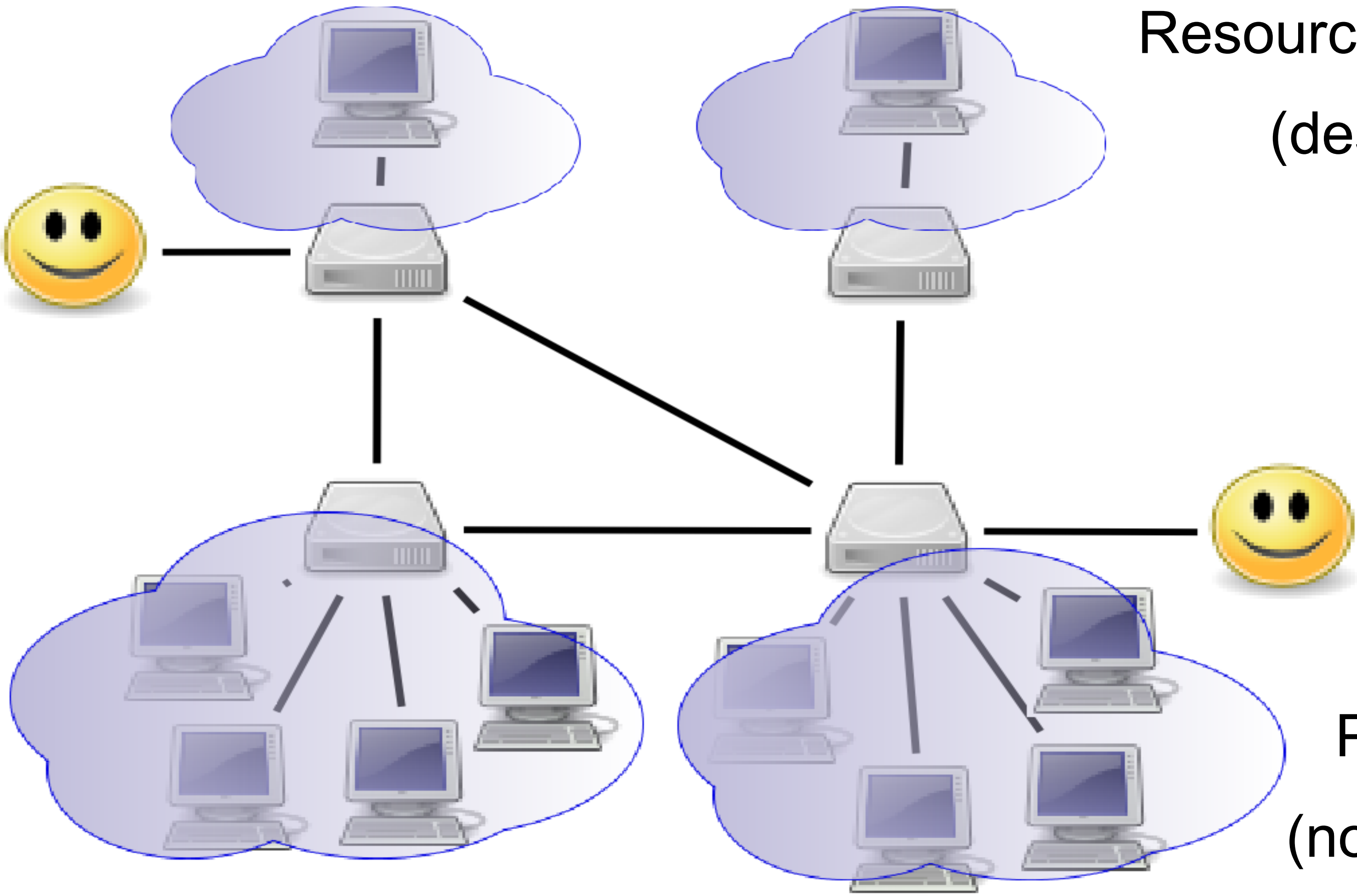
dynamic peering
relationships

opportunistic use of
additional worker nodes

graceful recovery
as worker nodes
become unavailable



Application model = Bag of Tasks
Grid model = Peer-to-Peer (2-levels)

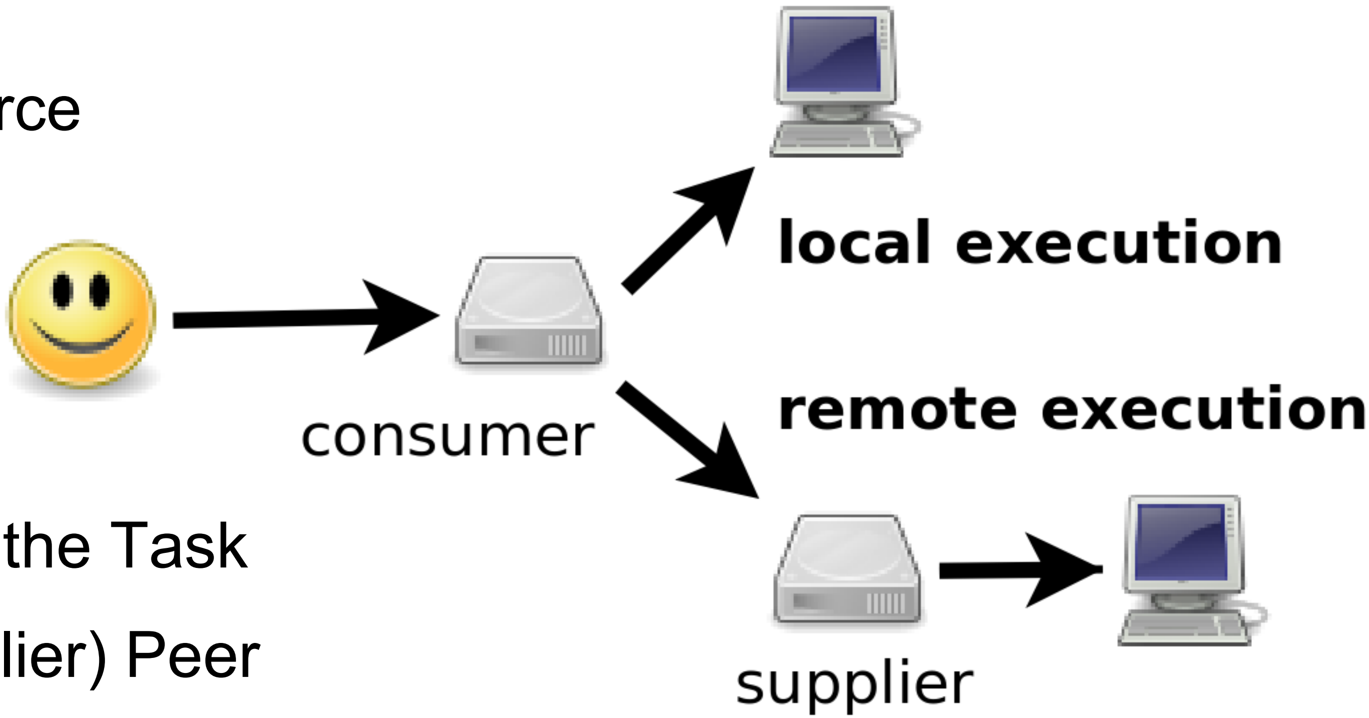


Resource = worker node (desktop computer)

Peer = controller (no privileged role, opaque to other Peers)

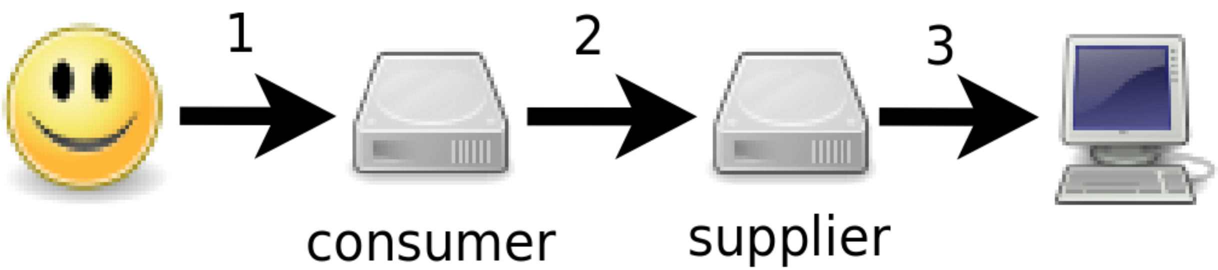
2 options to run Tasks

- send the Task to
  one local Resource

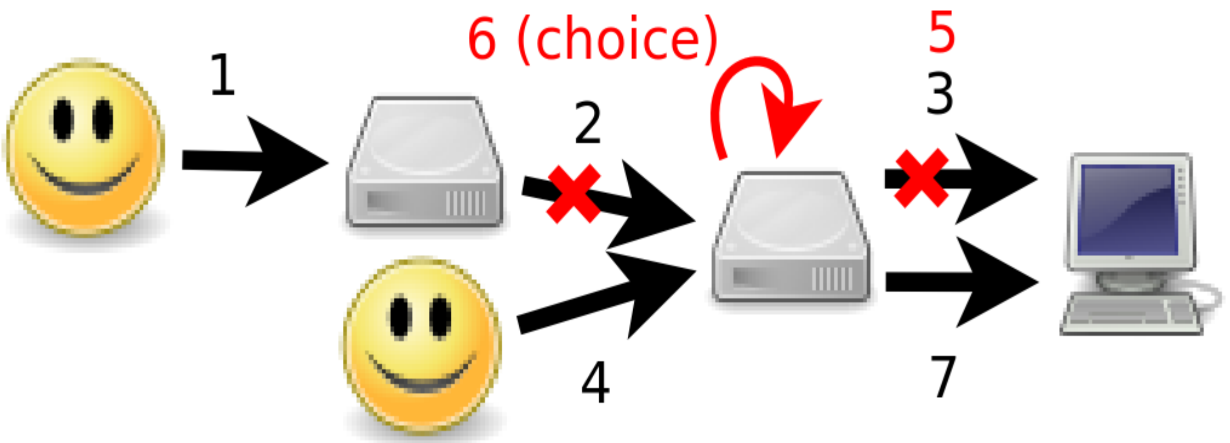


- (at peak) submit the Task
  to another (supplier) Peer

Task execution failures are frequent due to preemption



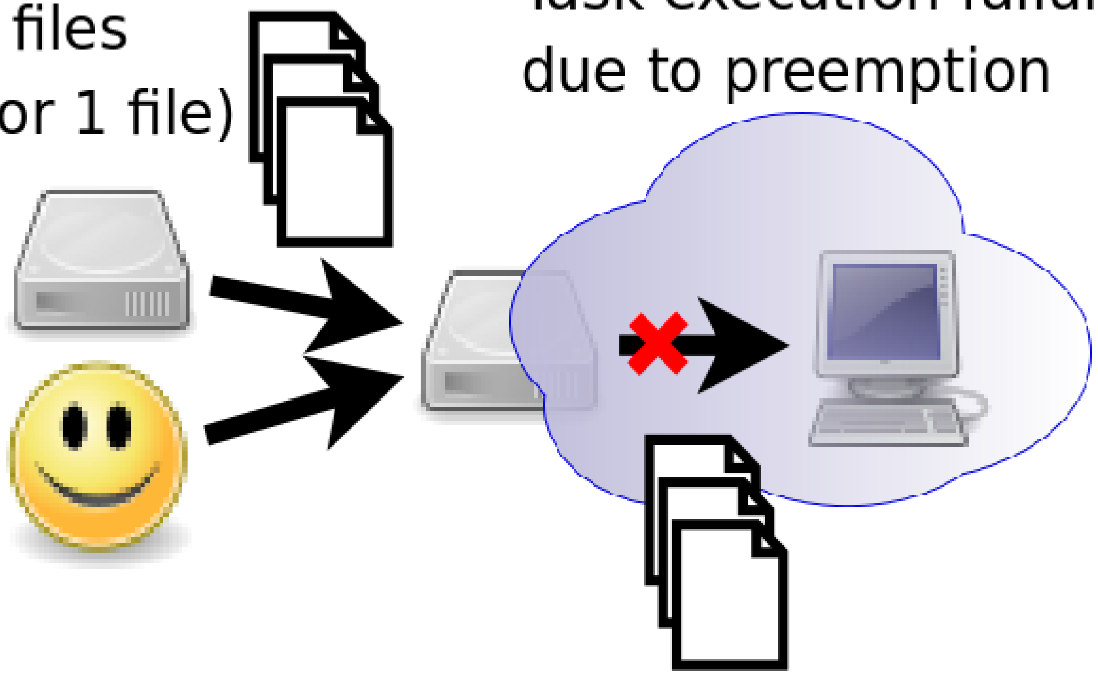
local use => preemption or cancellation => Task execution failure



Thesis objectives

repetitive patterns
in data files
(even for 1 file)

Task execution failures
due to preemption



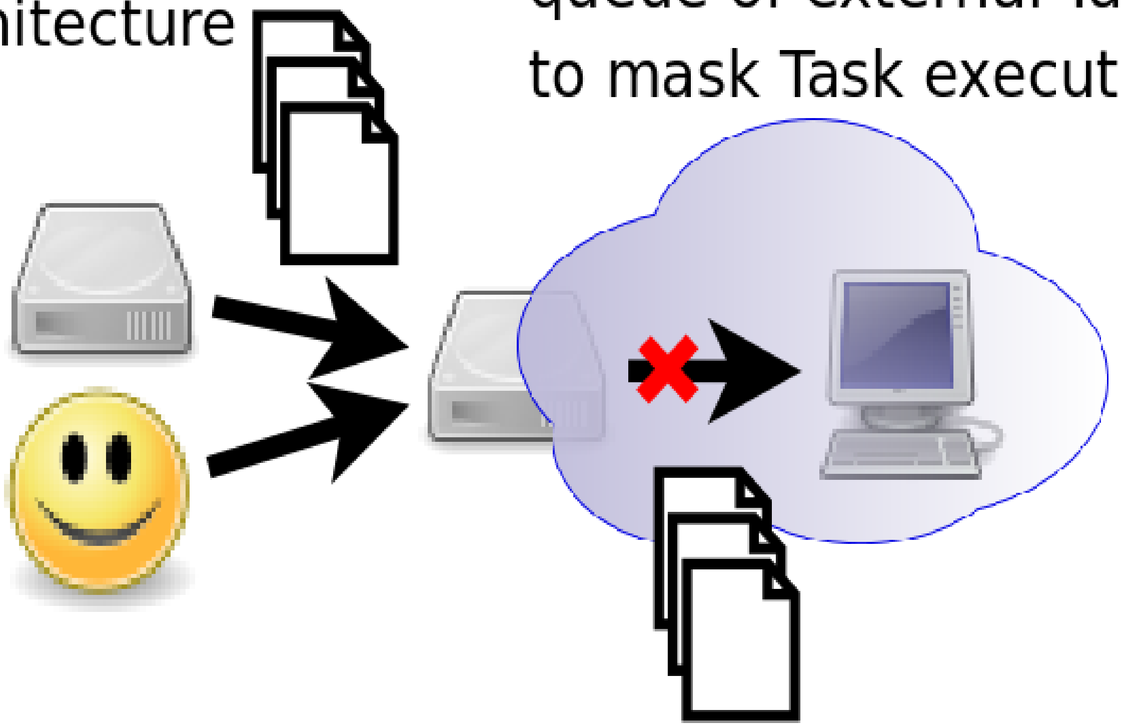
informational opacity
between Peers

reproducible tests challenging
due to distributed nature of the system

Thesis statement

P2P data transfer
architecture

queue of external Tasks
to mask Task execution failures



bartering:
decentralized sharing,
min. trust requirements

reproducible testing based on virtualization and simulation

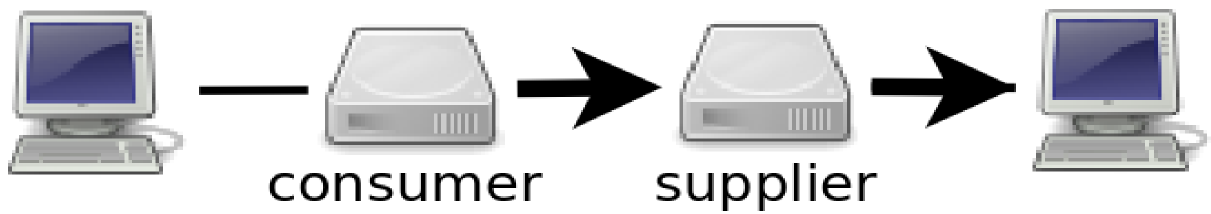
Lightweight Bartering Grid (LBG) middleware

Contents

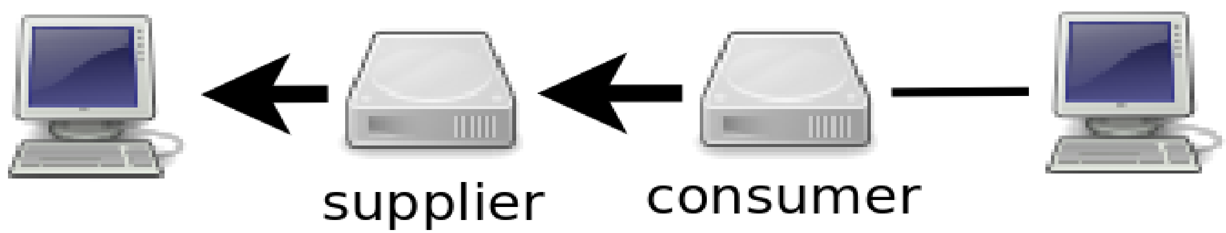
- Context & Thesis statement
- **Scheduling Tasks**
- Transferring large input data files
- Engineering P2P Grid software
- Running heavily-communicating Tasks
- Conclusion

Q: always reciprocate supplying?

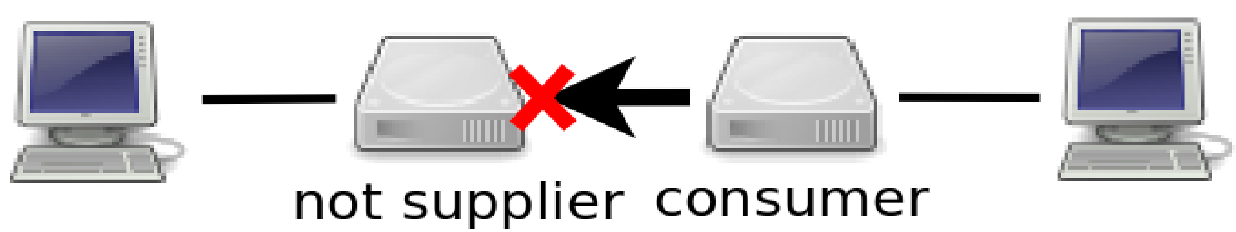
consume
computing time...



then reciprocate...



or not?



Take what you need, give what you do not need

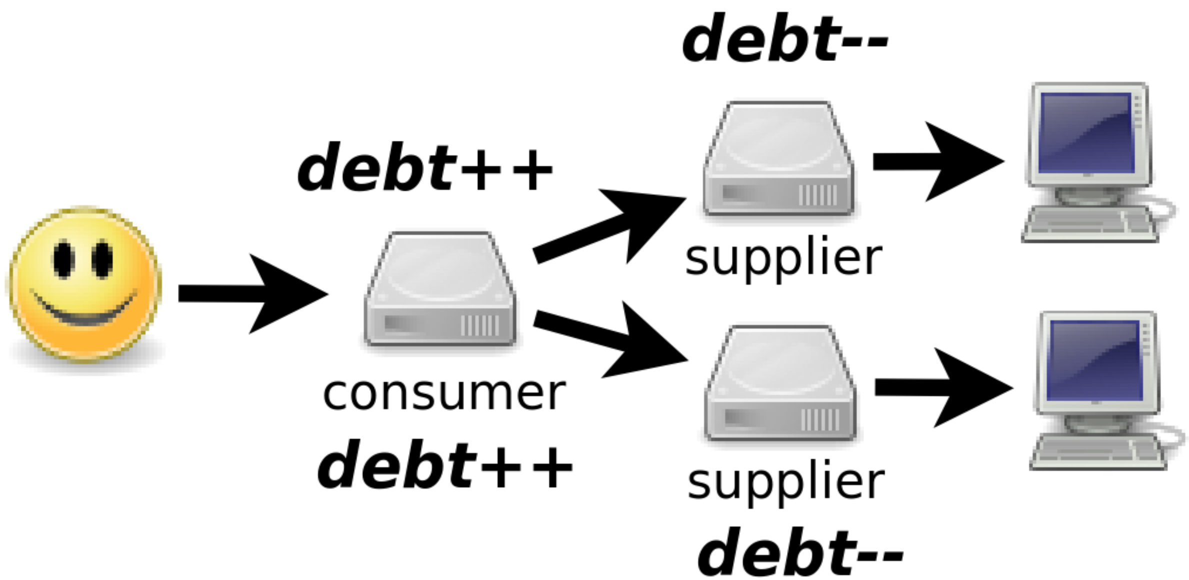
- Network of Favors model (state-of-the-art)
  - explains: when to supply, to which Peers
  - mitigates free riding

- basic behavior: always supply computing time of idle Resources
  even if no (recent) reciprocal consumption

- if several consumers want access to a Resource:
  supply to the Peer towards which most indebted

Each Peer tracks its own Grid usage

- Network of Favors = mechanism for fully decentralized bartering
- each Peer maintains its own accounting of
  « debts » of computing time, with each neighbor Peer



Bartering based on Network of Favors

- no guarantees, but opportunities of sharing when possible

- fully decentralized
  - preserves informational opacity between Peers
  - can be deployed today (no central banking component)

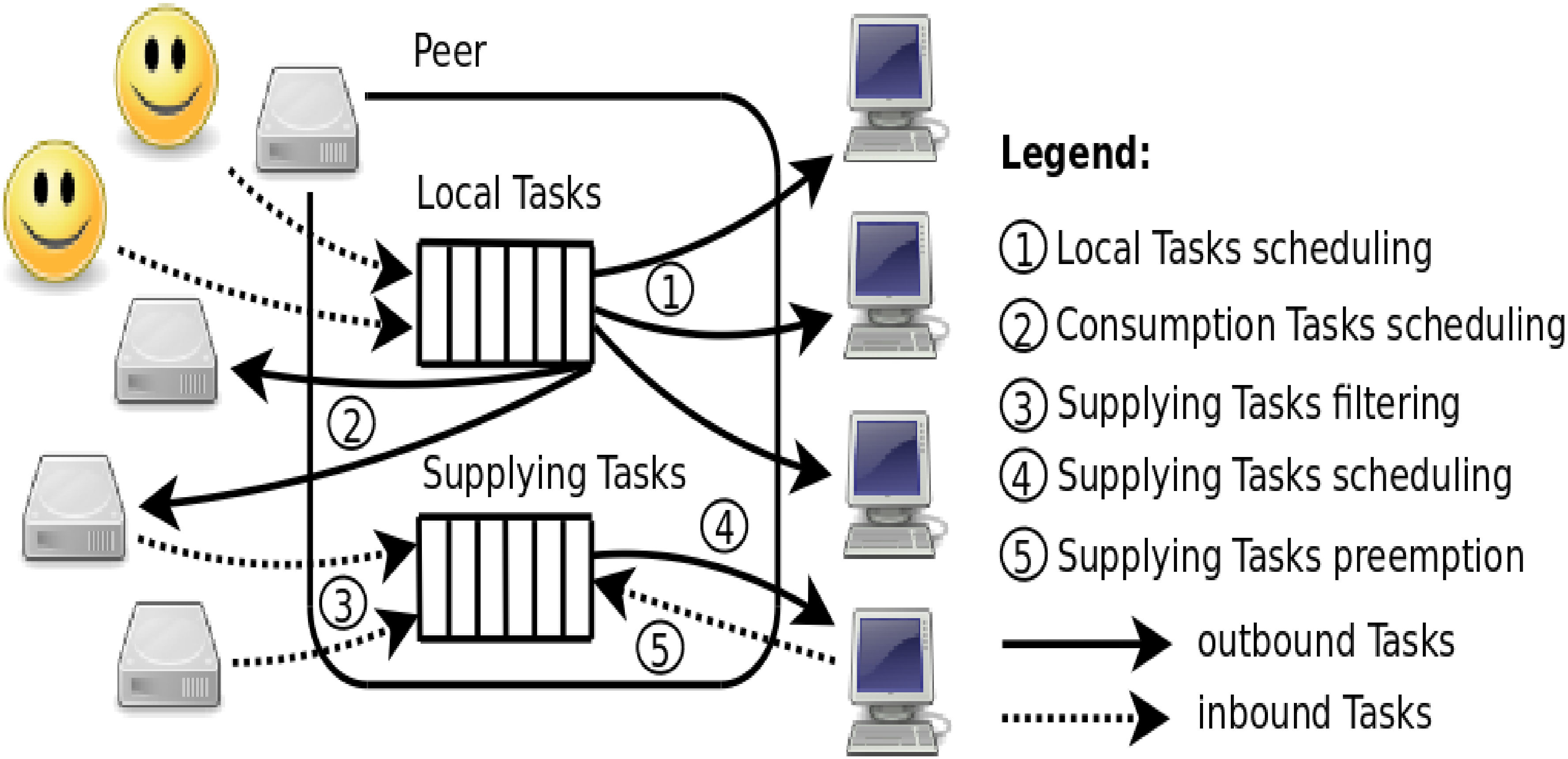
- **existing P2P Grids:**

  cannot hide Task execution failures to consumer Peers,

  because there is **no queueing support** for Supplying Tasks

Scheduling model

computations organized (Peer-level) around 2 Task queues:

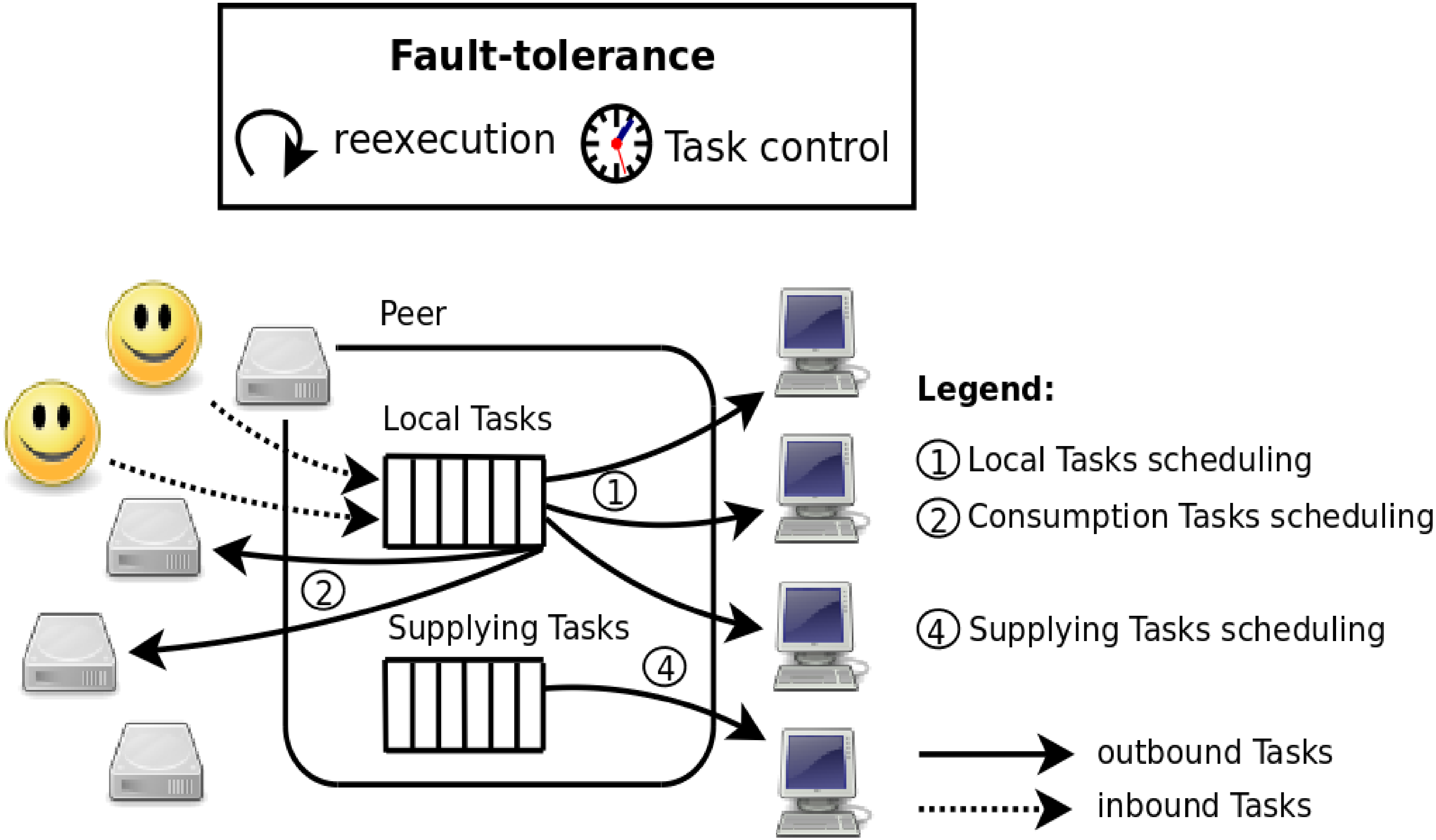


several “policy decision points” control the flow of Tasks

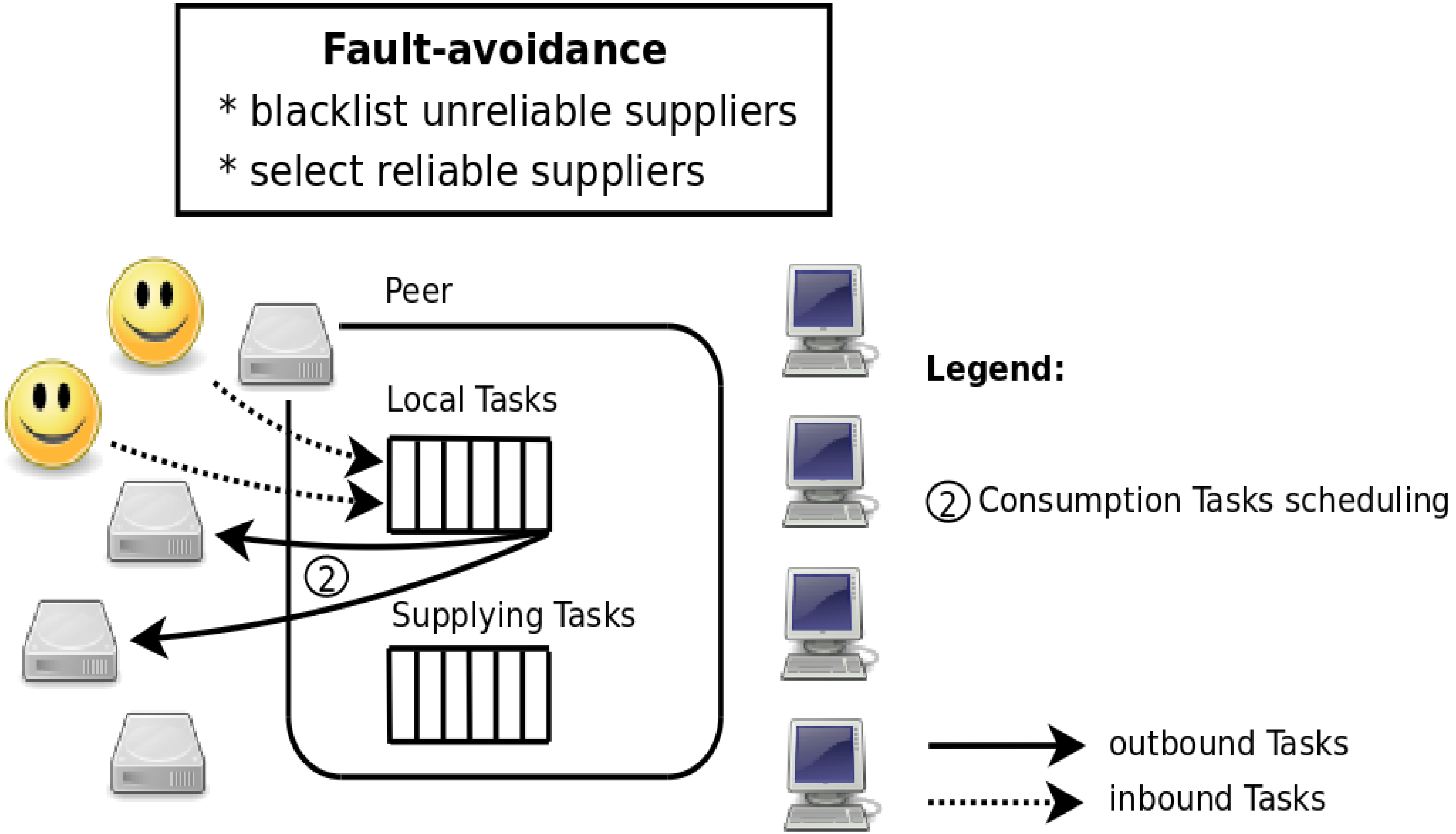
Fault-management classification

- fault-tolerance: gracefully adapt to faults after they happened
- fault-avoidance: avoid unreliable Peers
  (as a consumer)
- fault-prevention: avoid to cause faults to Tasks of other Peers
  (as a supplier)

Fault-tolerance mechanisms



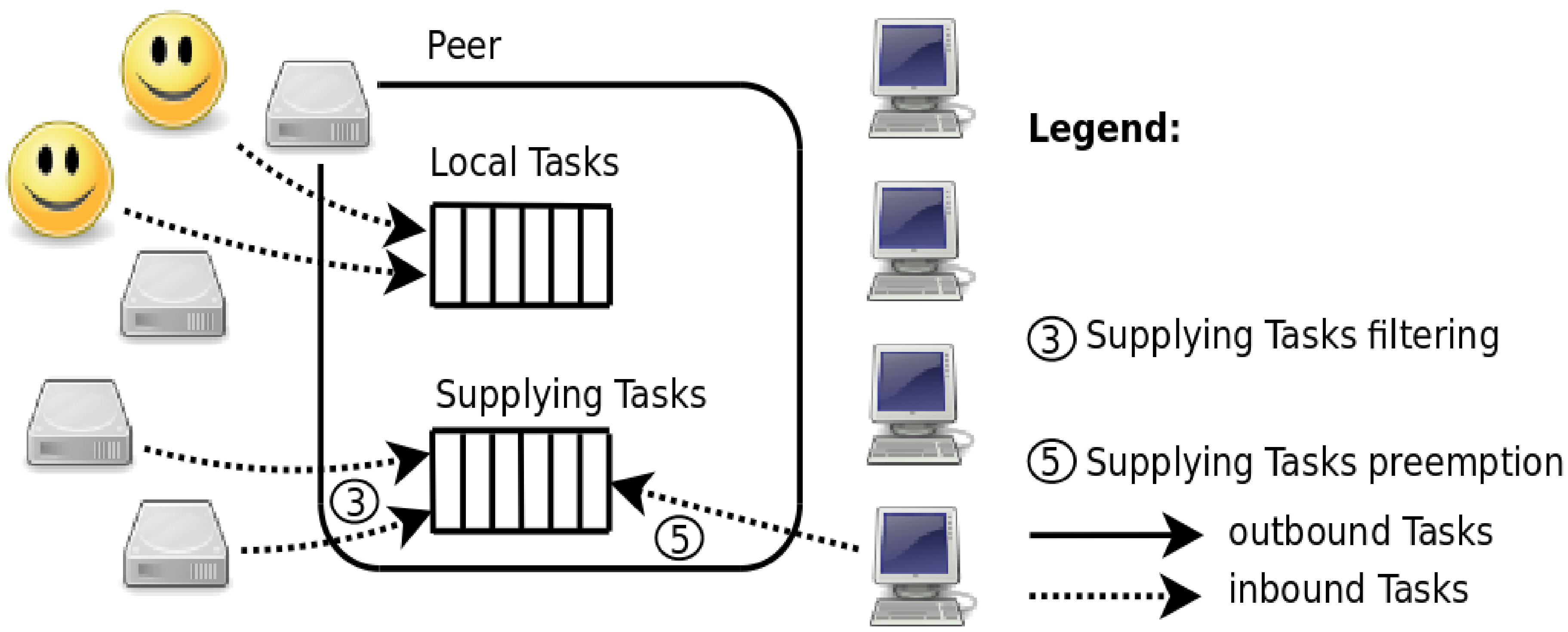
Fault-avoidance mechanisms



Fault-prevention mechanisms

Fault-prevention

* filter out Supplying Tasks to prevent long wait times
* adaptively preempt SupplyingTasks



Adaptive preemption and cancellation

behavior of a supplier Peer at peak, for fault-prevention:

- select for preemption the most recently scheduled Tasks

  i.e. who would “suffer” least (PSufferage heuristic)

- mask (preempt) or communicate (cancel) Task execution failure

  (cancellation lets consumer select another supplier)

- offer 2nd chance to long-running Tasks,

  with a short grace period

Contents

- Context & Thesis statement
- Scheduling Tasks
- **Transferring large input data files**
- Engineering P2P Grid software
- Running heavily-communicating Tasks
- Conclusion

Data transfers delay response times

- some Bags of Tasks process a large number of large files

  e.g. maps

- ... even implicitly

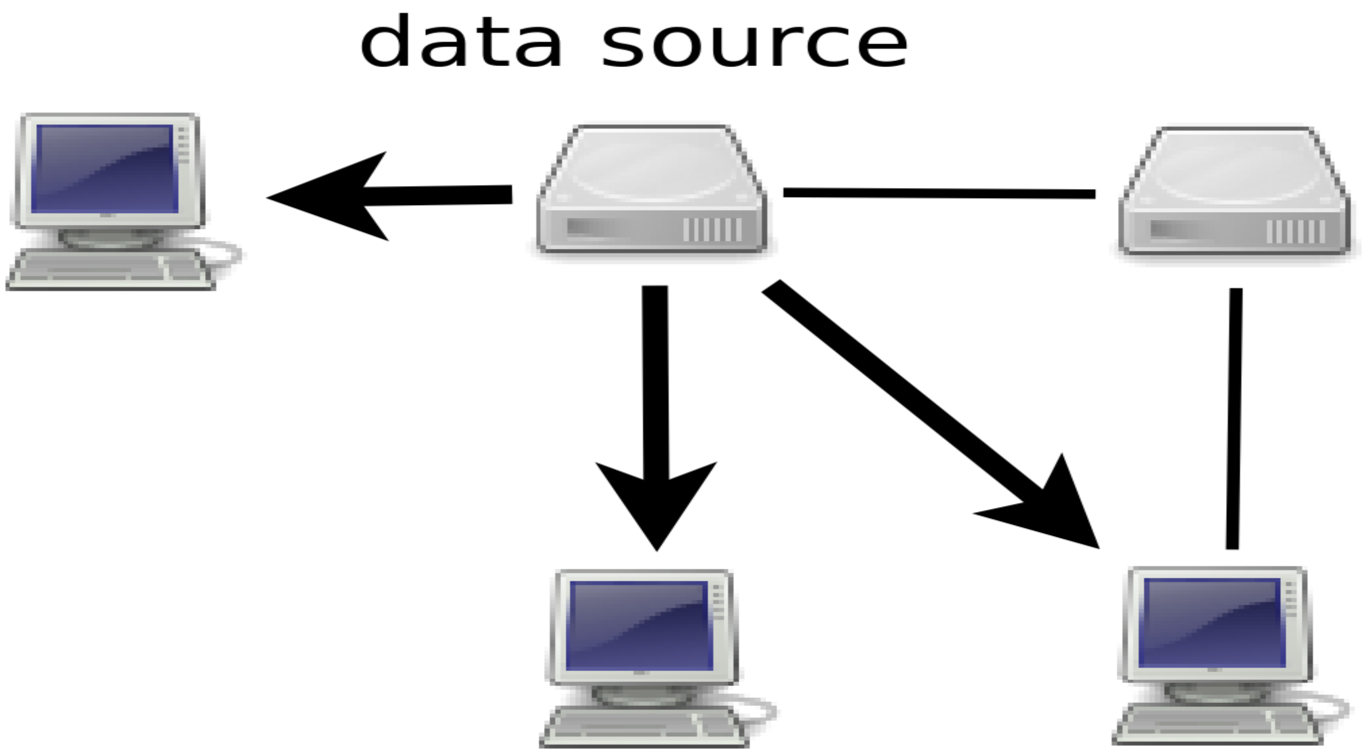
  e.g. so-called parameter sweeps

=>

- exploit (temporal, spatial) redundancy between data files

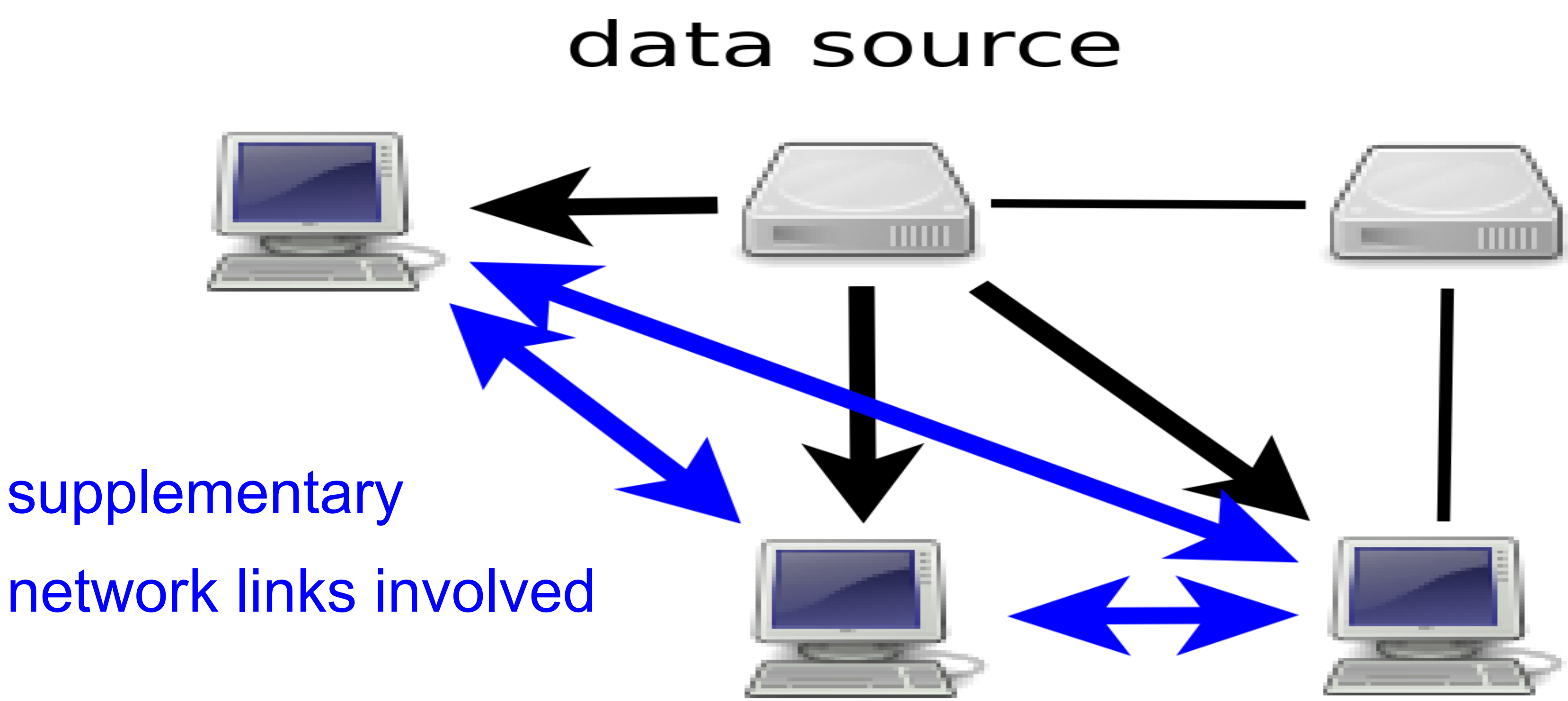
  to prevent unnecessary transfer costs

Centralized data transfers do not scale



P2P data transfers (e.g. BitTorrent) exploit orthogonal bandwidth

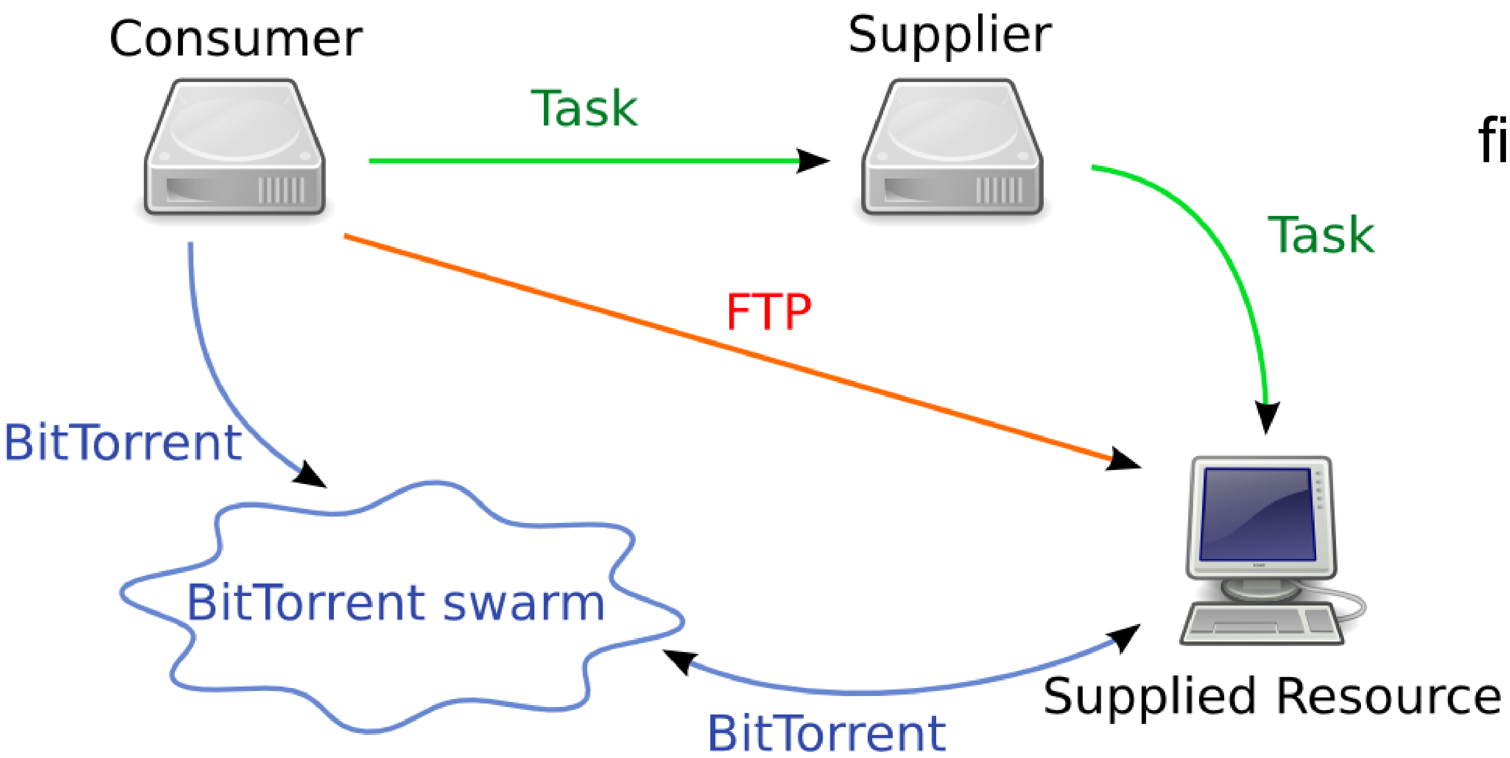
load spread between downloaders => reduced load on data source



time (N transfers of 1 file) ~ time (1 transfer 1 file)

Decentralized data transfer architecture

BitTorrent Nodes (= Grid Peers + Resources) **exchange** data



files transferred

with **FTP**

if < 50 MB

or # < 2

each Grid Peer runs its own BitTorrent **tracker**

Exploiting Temporal Data Redundancy

- Tasks with identical data files scheduled together

  (as simultaneously as possible)

- simultaneous transfers are initiated *on demand* (!)

  ... to maximize BitTorrent efficiency

P2P data transfers not always possible

- it may not be possible to schedule concurrently
  Tasks depending on identical data files
  (e.g. not enough Resources simultaneously available)

- some data files may be required
  by multiple Bags of Tasks spread over time

Exploiting Spatial Data Redundancy

- reuse data files to prevent unnecessary data transfers

distributed caching mechanism (each Resource)

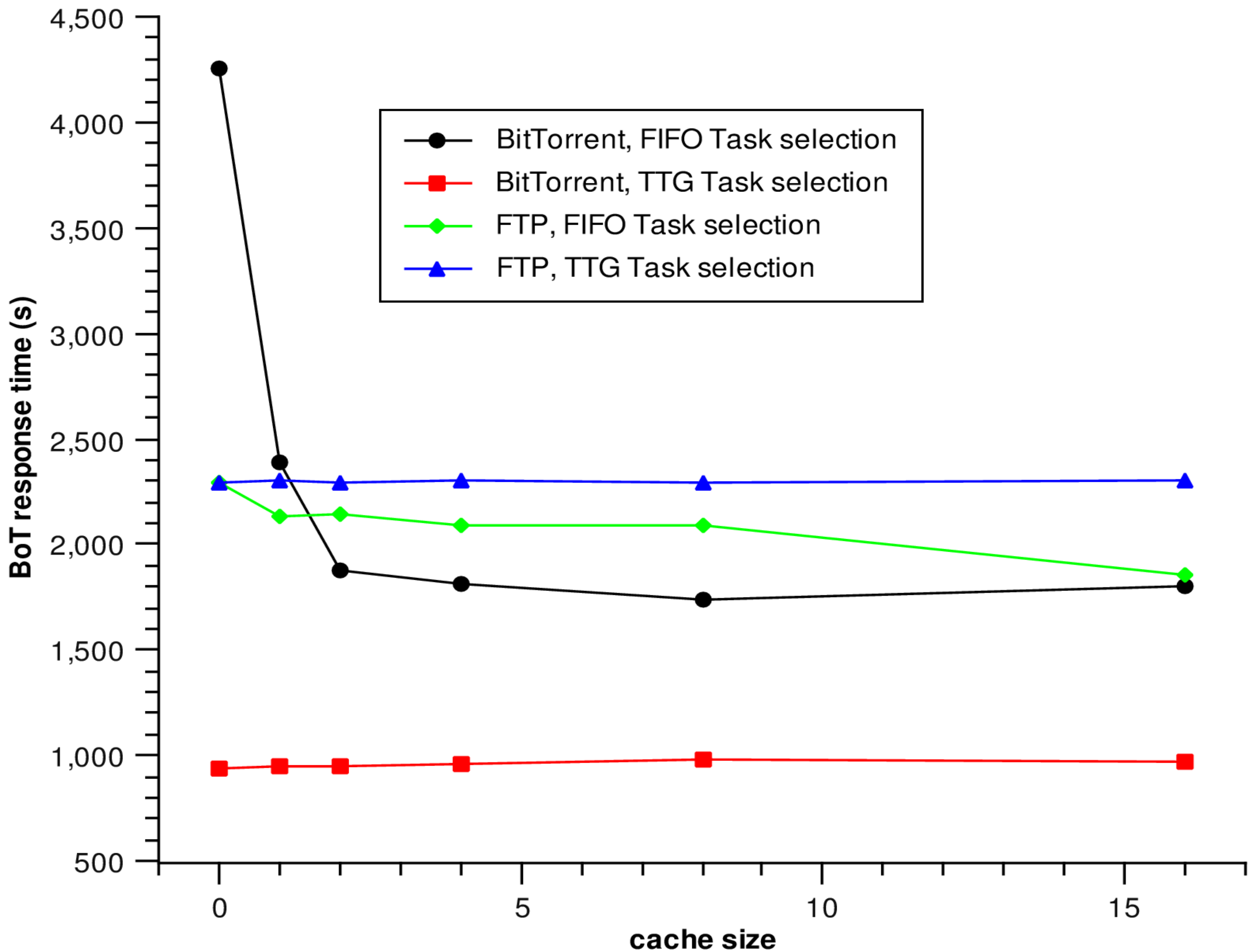
distributed data tracking mechanism (each Peer)

known for its Resources

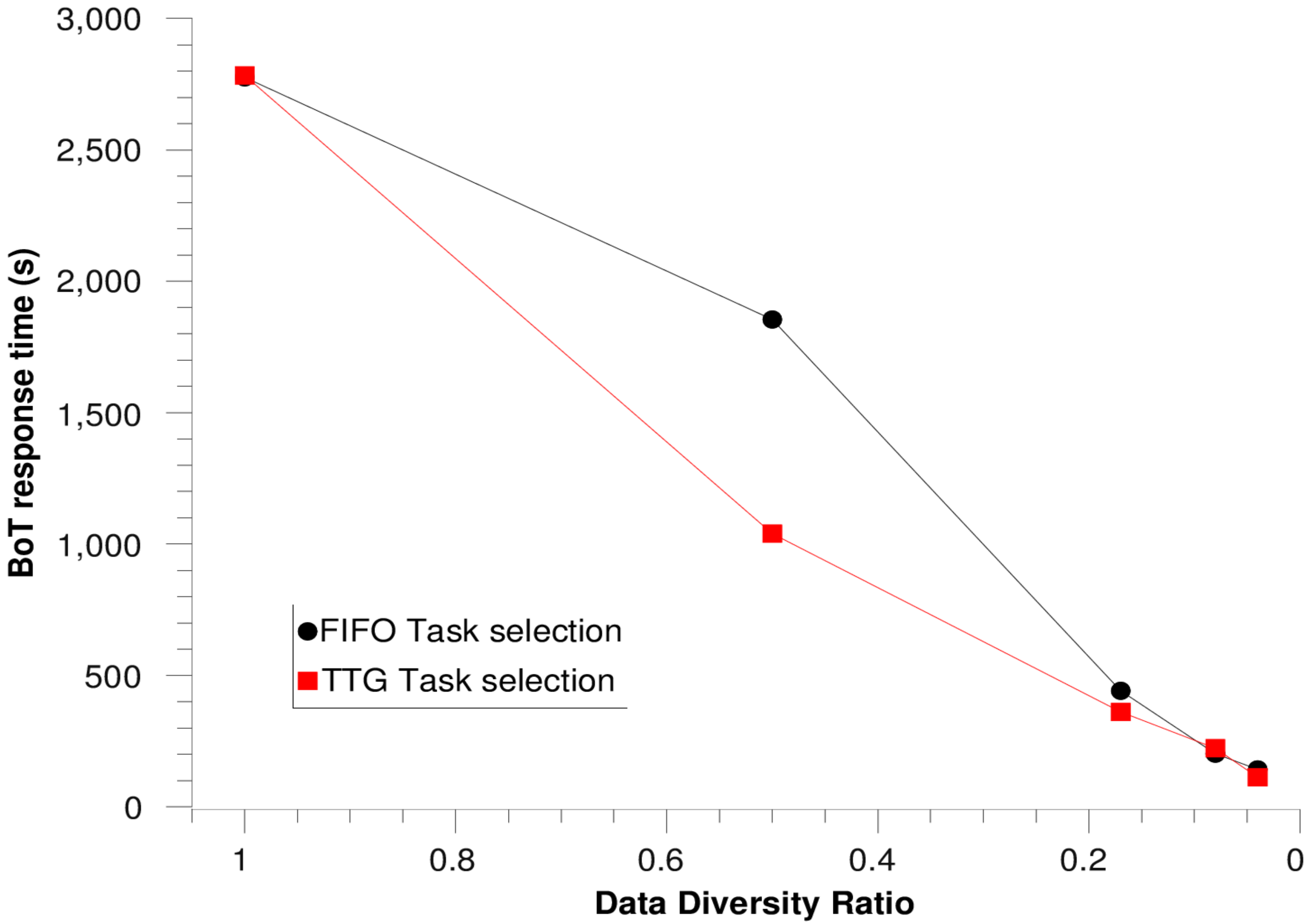
expected for recent suppliers

- data-aware scheduling to Resources, suppliers

256 MB file, 25x4 Tasks, 24 Resources BitTorrent vs. FTP, TTG vs. FIFO



256 MB file, 48 Tasks, 24 Res., BitTorrent variable redundancy, TTG vs. FIFO



Implicitly Exploiting Temporal Data Redundancy

- each Resource shares data files with BitTorrent
  even after they are not required anymore

- side effect of distributed caching:
  supplementary number sharing sources
  =>
  implicit Temporal Tasks Grouping
  =>
  load removed from the data source with BitTorrent

Summary of data redundancy exploitation

- BitTorrent (Temporal Task Grouping)

  if parallel execution & data transfer both possible

- distributed caching + data-awareness (Spatial Task Grouping)

  if parallel execution not possible &

  if data available on idle Resources

- BitTorrent + distributed caching (implicit Temporal Task Grouping)

  if parallel execution not possible &

  if data not available on idle Res. (i.e. available on busy Res.)

Contents

- Context & Thesis statement
- Scheduling Tasks
- Transferring large input data files
- **Engineering P2P Grid software**
- Running heavily-communicating Tasks
- Conclusion

Testing P2P Grid software is complex

- multiple sources of bugs: large software, scheduling algorithms, state consistency, network, code execution, multithreading, data transfers, ...

- difficult to set a P2P Grid into a given state
  because P2P Grid = complex, non-dedicated, distributed

- virtualization of messaging
  =>
  virtualized execution in a controlled environment

Virtualization alone is not scalable

- 24 hours of virtualized execution = 24 hours

  ... not temporally-scalable (i.e. execution occurs in real time)

- also virtualize time-consuming operations

  i.e. simulate Task execution, timers, multithreading

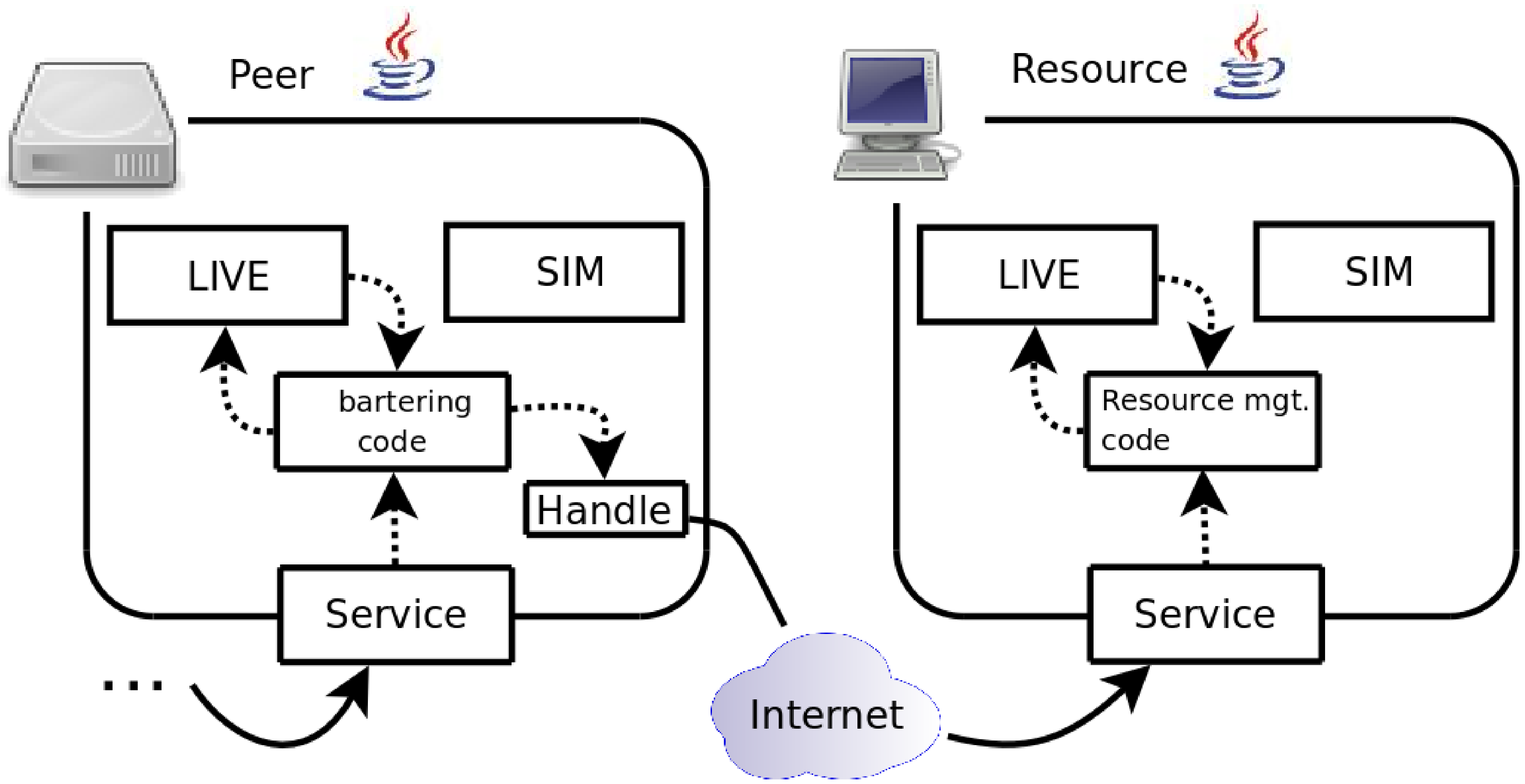
- discrete-event simulation can enable reproducible evaluations

  ... but simulation accuracy often limited

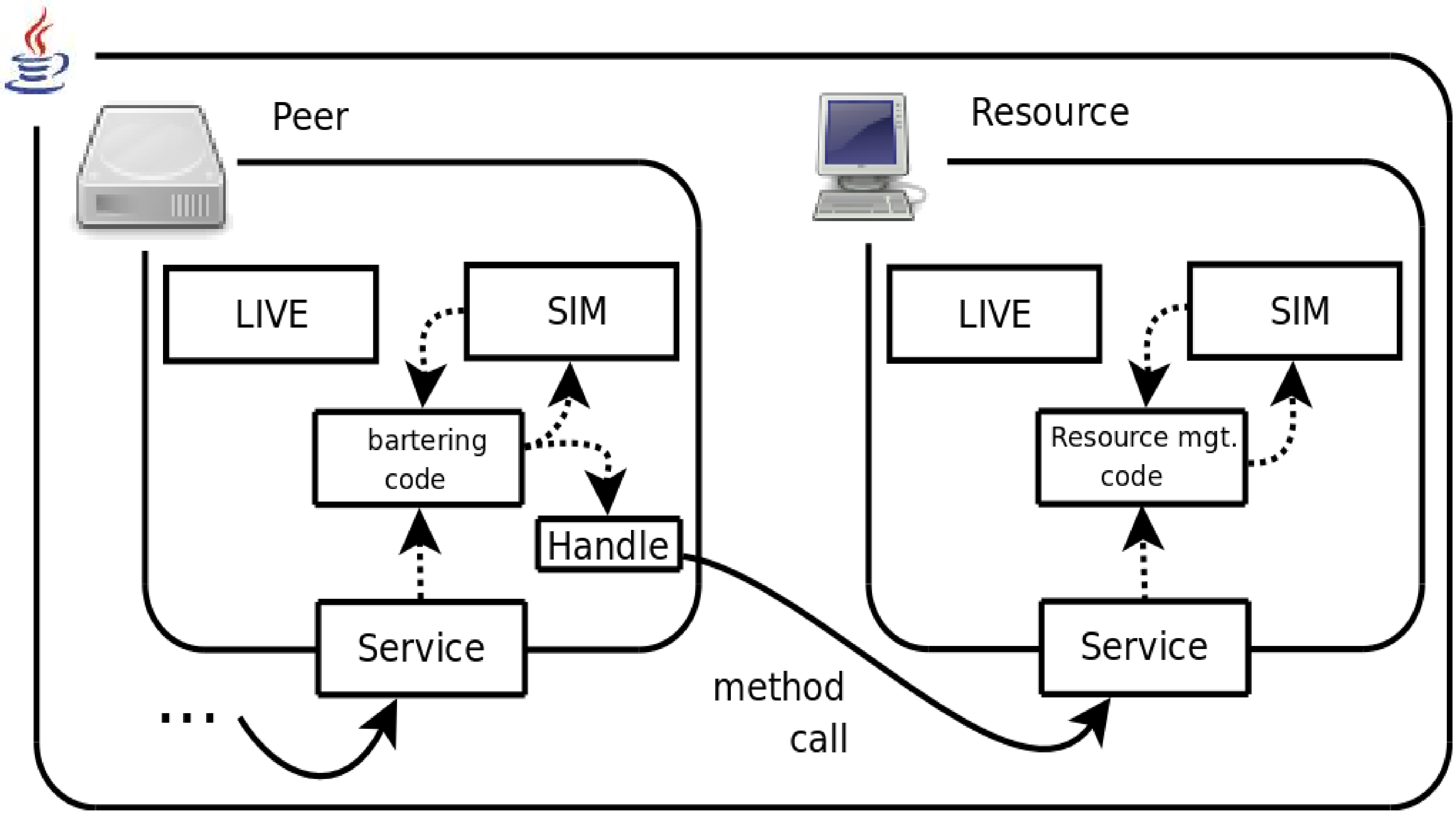
Code once, deploy twice (Grid Reality And Simulation, M. Quinson)

- idea: virtualization + simulation = software engineering tool
- STEP 1: completely virtualize Grid nodes at middleware-level, i.e. ~~Virtual Machine (e.g. Xen, VMWare)~~, ~~O.S.-level emulation~~
- STEP 2: then weave simulator code with scheduling algorithms
- massive code reuse between implementations: first, top-down application of *code once, deploy twice* to a complete middleware

Communications in the middleware



Communications in the simulator



Simulator overview

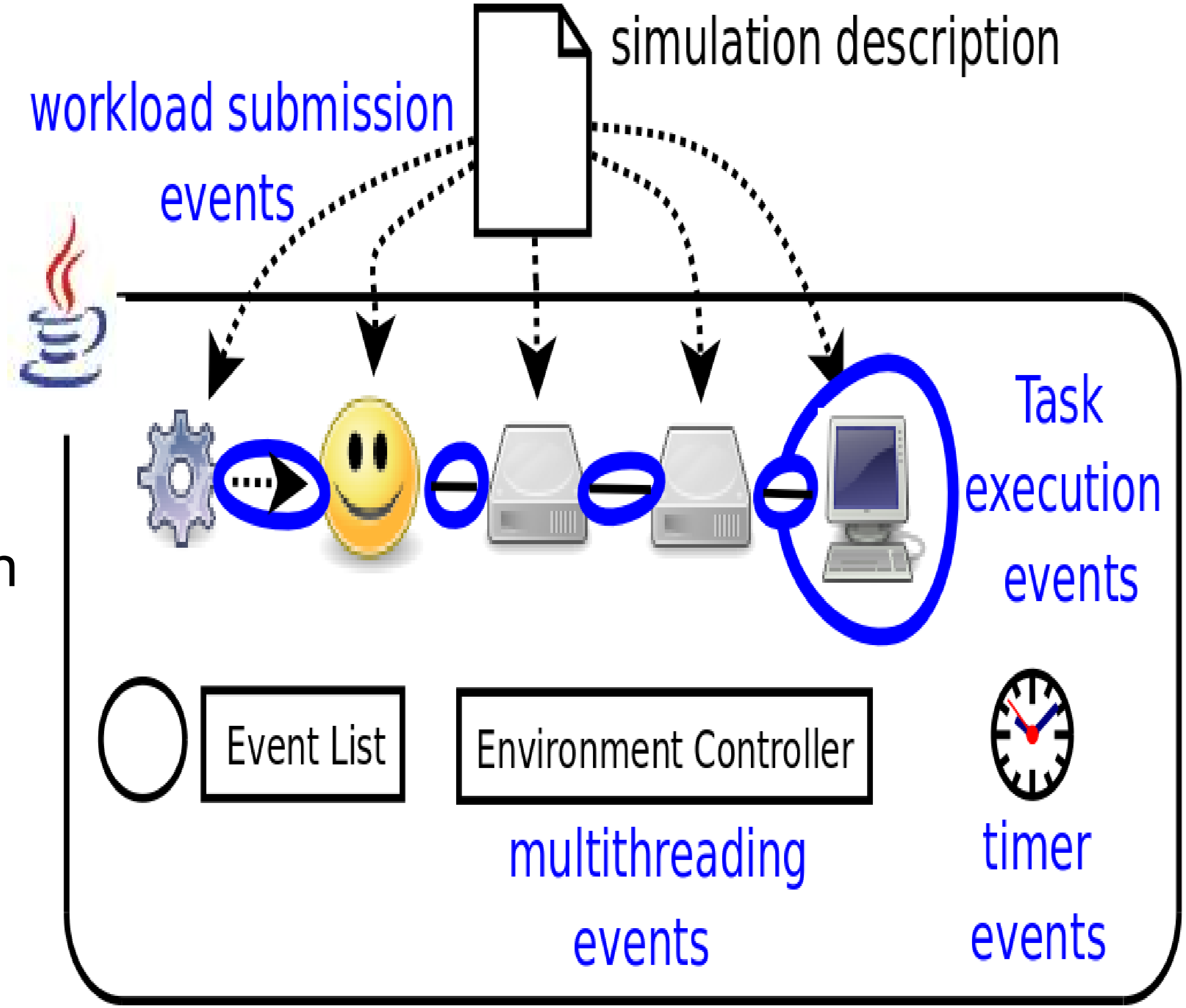
simulation language

input file:

- Grid topology
- synthetic workload
- Peers configuration

output file:

- execution stats



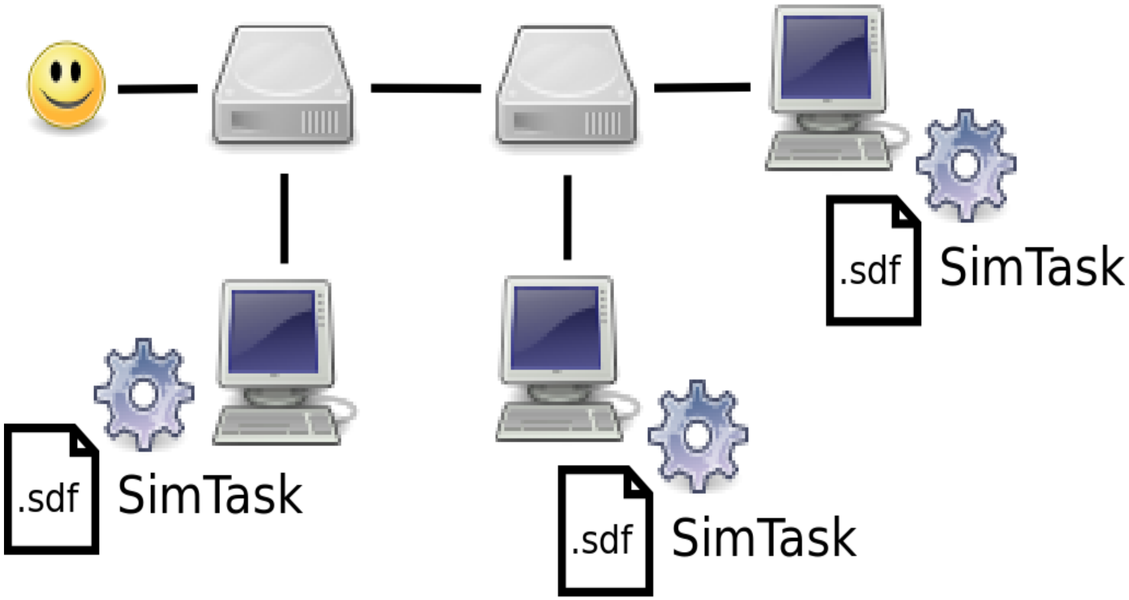
Reproducible testing

practical benefits:

- issues with live deployment replayed in the simulator
- most of the code tested before going live, <u>at high speed</u>
- simulated algorithms deployed <u>as-is</u> in the middleware
- large-scale parameter sweeps of scheduling policies

Self-Bootstrapping

- self-bootstrapping = current, stable version of a given system used to develop next version
- Bag of SimTasks (N simulators embedded into Grid Tasks)
- 1 middleware:
  basic policies
- N simulators (SimTasks):
  test and evaluate
  advanced policies



Contents

- Context & Thesis statement
- Scheduling Tasks
- Transferring large input data files
- Engineering P2P Grid software
- **Running heavily-communicating Tasks**
- Conclusion

LaBo Grid
Lattice-Boltzmann computations on a Grid

G. Dethier's research,

with Chemical Engineering dept.:

Computational Fluid Dynamics

simulation of flows on a lattice

with Lattice-Boltzmann method

Iterative Stencil application

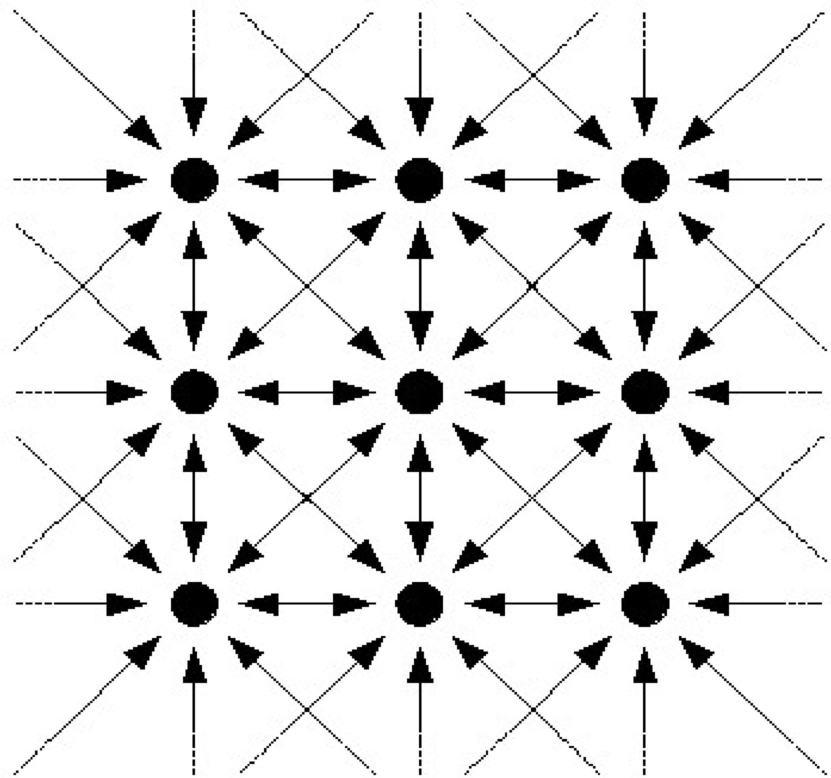
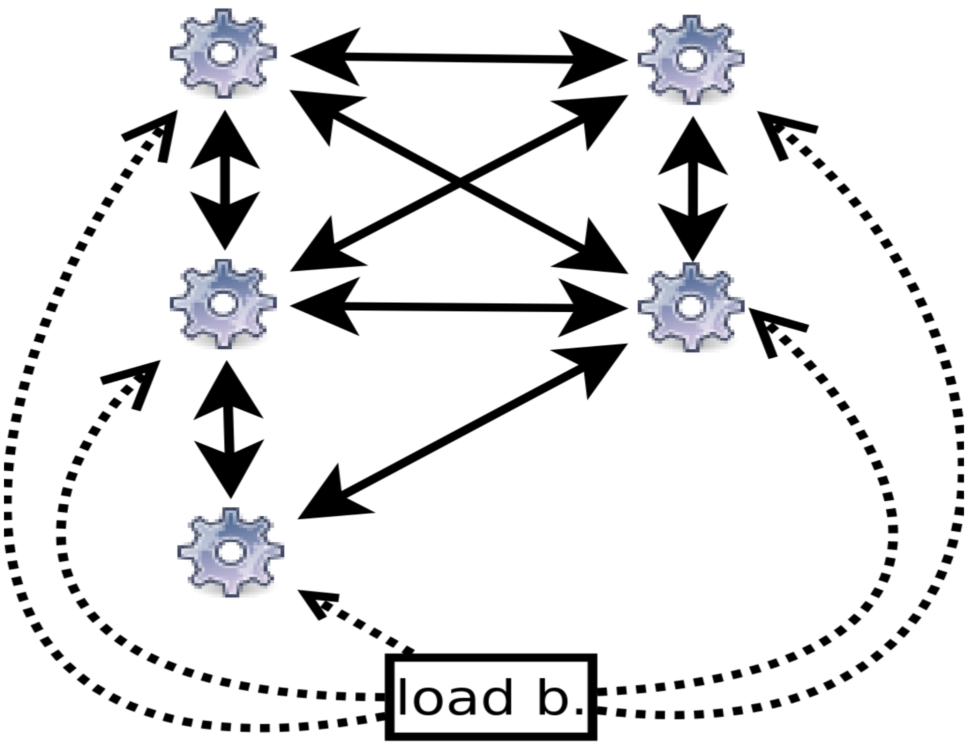


Figure courtesy of Gérard Dethier

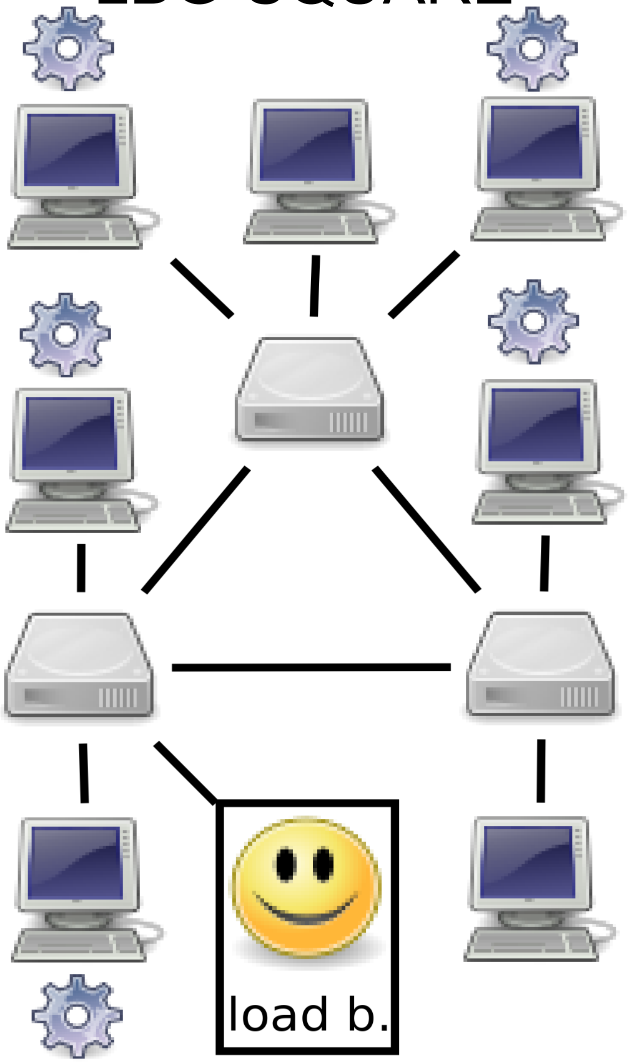
LBG-SQUARE = LBG x LBG
(Lattice-Boltzmann on the Grid x Lightweight Bartering Grid)

LaBo Grid



(currently centralized) load balancer

LBG-SQUARE



Locality-aware co-allocation

- how to balance load in a P2P Grid?
  - dynamic large-scale co-allocation
  - ... thus no advance knowledge of Task schedule

  => no way to mold Tasks before deployment

- load balancing in LaBo Grid performed after scheduling:

  dynamic benchmarks

  (Gérard Dethier's work on adaption to CPU and network cap.)

  => co-allocation by P2P Grid, locality-awareness by LaBoGrid

Fault-tolerance with checkpoint-restart

- **challenge: decentralized architecture for scalability**

  => **P2P** checkpointing and fault recovery

  - distributed checkpoint storage, transfer and reload

    (i.e. no centralized checkpoint server)
  - nominal operations, checkpoint reload = decentralized
  - load (re)balancing = (currently) centralized

- **challenge: bursts of Task execution failures (preemption)**

  => **P2P-aware** checkpoint storage

  i.e. checkpoints of 1 Task spread to different Peers

Contents

- Context & Thesis statement
- Scheduling Tasks
- Transferring large input data files
- Engineering P2P Grid software
- Running heavily-communicating Tasks
- **Conclusion**

Contributions

- scheduling model with queueing support,
  systematic review of possible policies
  (proposal of a new efficient one: adaptive preemption)

- P2P data transfer for P2P Grid computing
  - BitTorrent (TTG)
  - distributed caching + data-awareness (STG)
  - BitTorrent + distributed caching (implicit TTG)

Contributions (continued)

- software engineering
  - first, top-down application to a complete middleware of *code once, deploy twice* (Grid Reality And Simulation, M. Quinson)
  - reproducible testing

- execution of Iterative Stencils
  - LBG-SQUARE (locality-aware co-allocation)
  - P2P-aware P2P checkpointing mechanism (fault-tolerance)

Perspectives

Scheduling

- investigate Task replication as well as reservations
- simulation of data transfers, better simulation of multithreading
- measure system-wide impact of local scheduling choices

Middleware scalability

- improve even more the scalability of data transfers
  (CDN-like data replication, adaptive data compression)
- large-scale deployment (Cloud Computing, Volunteer Grid)

Thank You.