



Workshop on Models, Algorithms and Methodologies for Grid Computing Environment

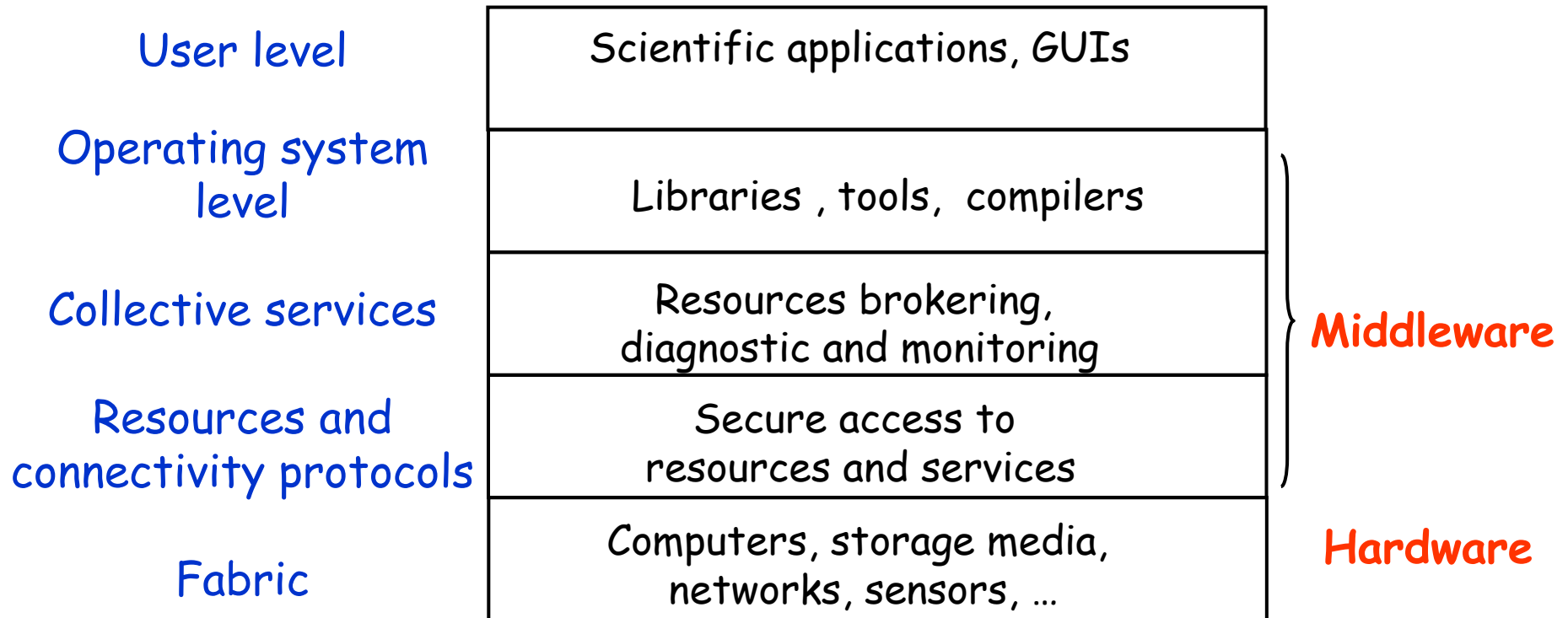
## Implementing effective data management policies in distributed and grid computing environments

Luisa Carracciuolo, Giuliano Laccetti and Marco Lapegna

Italian National Research Council  
and  
Univ. of Naples "Federico II" - Italy

This work is partially supported by Italian Ministry of  
Education, University and Research within the activities of the SCOPE project

# Grid infrastructure



Common middleware tasks:

- Resources discovery and allocation
- Grid monitoring
- Performance contract definition
- ...

# Challenging scientific applications

---

- very large input data
- dependencies among subproblems

e.g. fluid dynamic codes



It is not sufficient to choose the most powerful resource in the grid



It is necessary

- To define suitable methodologies to distribute data onto the grid components
- To overlap communications and computations
- To provide tools that eliminate unnecessary data transfers

# Dominant style in parallel computing

---

- **Single Program Multiple Data**
  - All nodes use the same program on different data
  - Effective tools available (eg. MPI)
  - Need frequent data communications and synchronizations in a systolic fashion
  - Notable example: ScaLAPACK project



Optimal for static and dedicated computing environments

# A suitable programming model

---

Grid computing  
keywords

- Sharedness
- Heterogeneity



SPMD programming model  
is impracticable

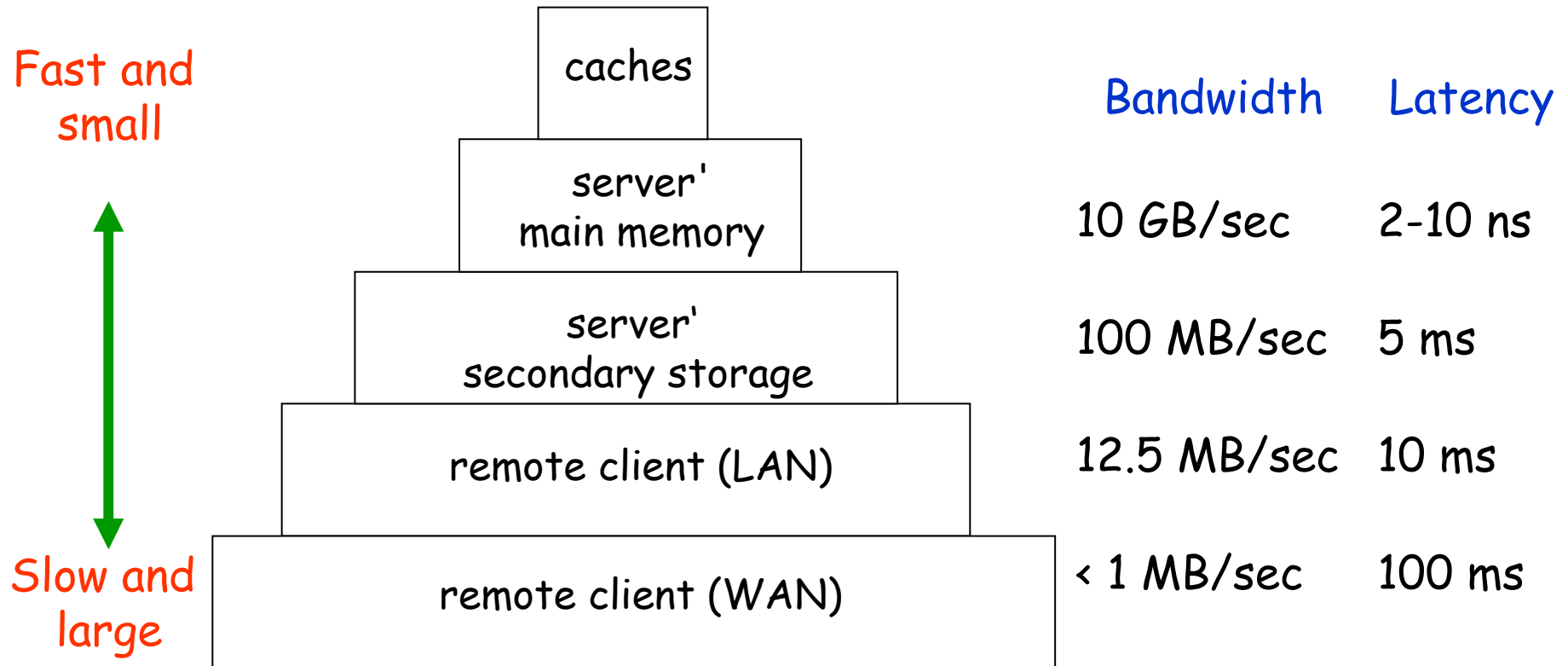
Need of a programming  
model without  
communications and  
synchronization among  
the computing nodes



Client / server  
programming  
model

Notable example: SETI@home

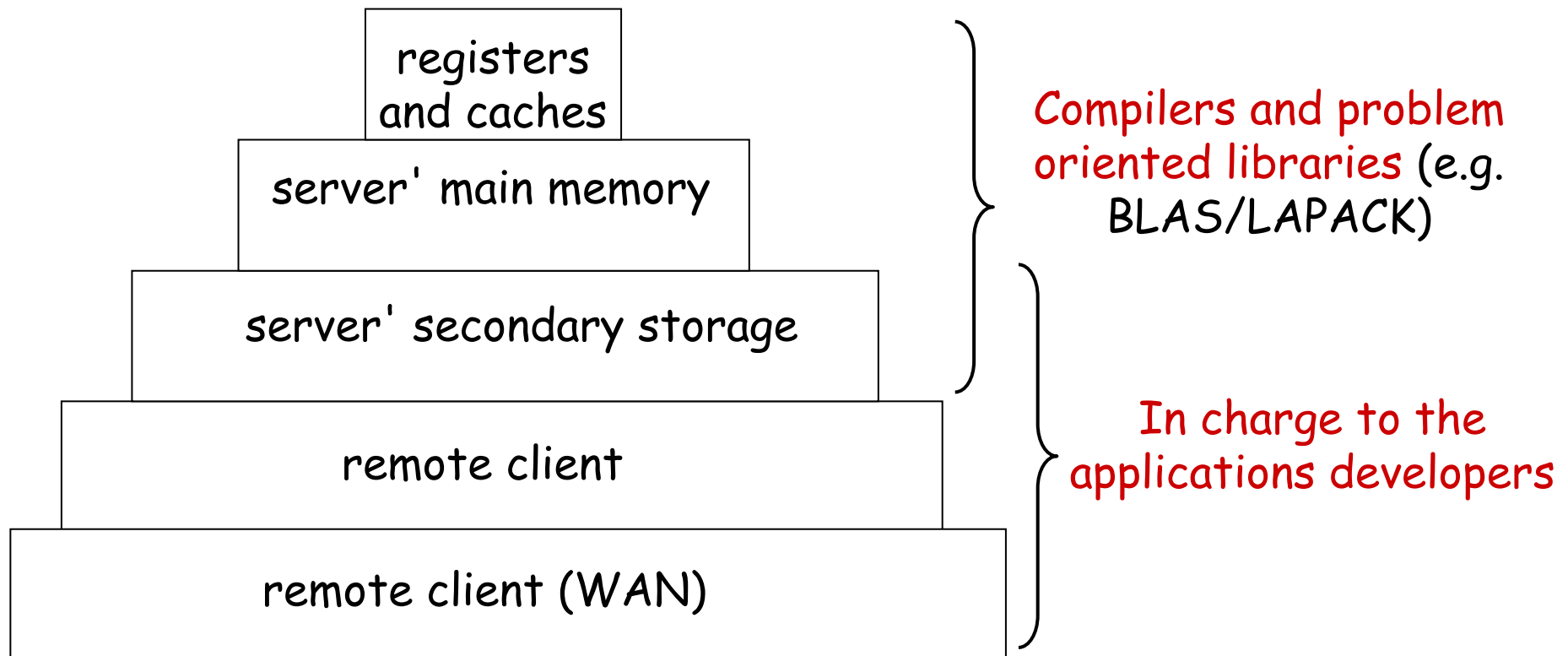
# A memory-aware model of a grid system



use ideas and methodologies for NUMA machines and to extend them to the new environment

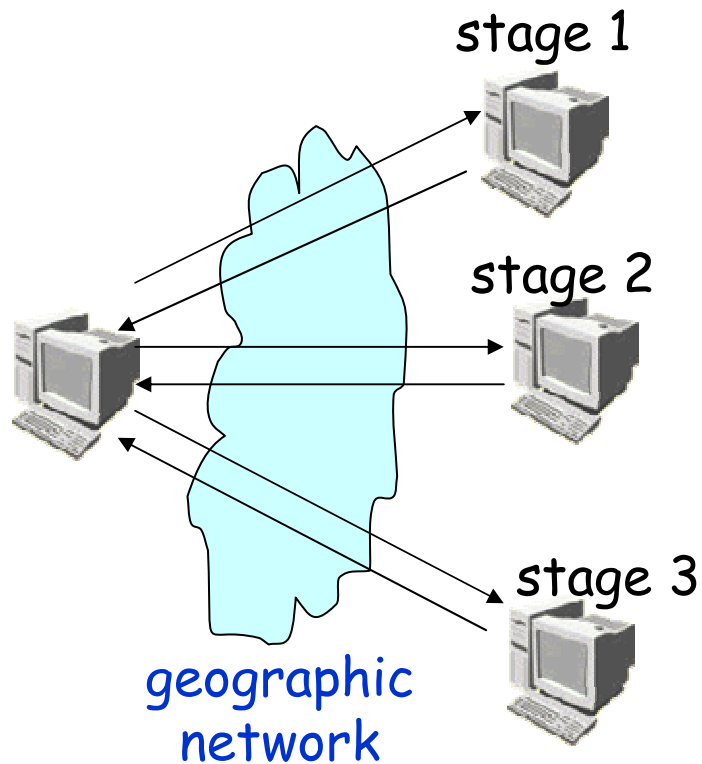
# Key strategy for NUMA machines

Extensive use of caching strategies  
at each level of the memory hierarchy

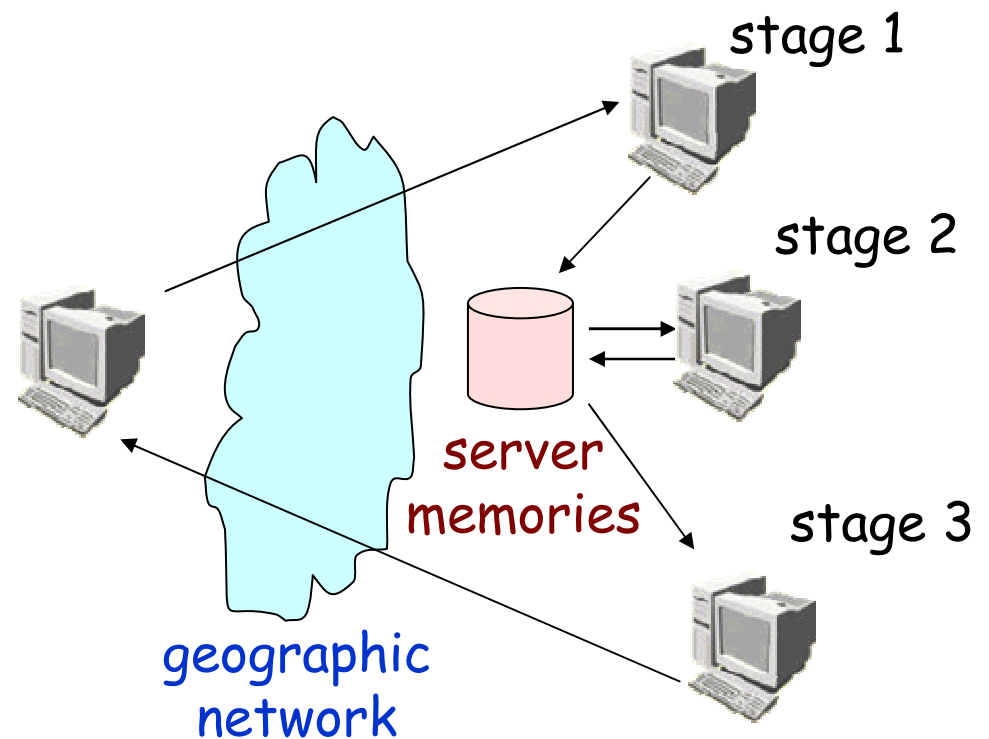


# Example:

## Three pipelined stages in a client/server environment



No cache and superfluous network traffic



Use of server memories as cache and comm/comp overlapping

## Two options for the cache

---

- **Server main memory**
  - Fast but small
- **Server secondary storage**
  - Large amount of available space
  - Access time similar to that of the server main memory (respect to the access time to the client)



**Our choice for the cache:**  
server secondary storage

## Environment we used: NetSolve 2.0

---

- One of the best known middleware for grid computing
- Based on a client/agent/server paradigm
  - The agent selects the most suitable server on the basis of the performance of the servers
  - The client sends and receives data directly from the server
  - The server performs the computation using the legacy software interfacing with the environment by means of a Problem Description File

The data exchange is executed for every request to NetSolve



Superfluous network traffic in case of pipelined stages

# Tools to manage data in NetSolve:

---

## Request Sequencing

- NetSolve construct that defines a Direct Acyclic Graph
- Nodes represent tasks and arcs represent dependencies
- **Main limitation:** The entire DAG sequence is executed by only one server (even if there are independent tasks)

## Distributed Storage Infrastructure

- Infrastructure allowing the client to place data in a storage accessible by the server
- Currently implemented by means of the Internet Backplane Protocol (IBP)
- **Main limitation:** The servers are able to read data from the IBP storage, but they appear to be unable to write on it

# Tools to manage data in NetSolve: 2

---

## Distributed Storage Infrastructure

- Infrastructure allowing the client to place data in a storage accessible by the server
- Currently implemented by means of the Internet Backplane Protocol (IBP)

### Main limitation:

The servers are able to read data from the IBP storage, but they appear to be unable to write on it

# Our task: to modify the NS DSI infrastructure

---

- DSI infrastructure defines the new data type DSI\_OBJECT to describe the features of the IBP storage
- Among the fields in DSI\_OBJECT there are the read/write/manage capabilities (unique character strings used as keys to access the IBP storage)
- The DSI\_OBJECT are generate by the client and sent to the server by means the NS infrastructure,

# Our task: to modify the NS DSI infrastructure

---

## THE PROBLEM

the **legacy software cannot call directly the DSI functions** because the Problem Description File used to generate the server code is unable to manage the `DSI_OBJECT`

- Our **changes** to the DSI infrastructure are **aimed to use the DSI functions directly from the server legacy software**
- In the modified DSI infrastructure **the APIs of the DSI functions include the read/write/manage capabilities** (character strings) **in place of the `DSI_OBJECT`**, so that they can be used also in the server legacy software

# Example

---

Old DSI API:

```
int ns_dsi_read_vector( DSI_OBJECT* dsi_obj, void* data,  
int count, int data_type)
```

Modified DSI API:

```
int ns_new_read_vector( char* ibp_read_cap, void* data,  
int count, int data_type)
```

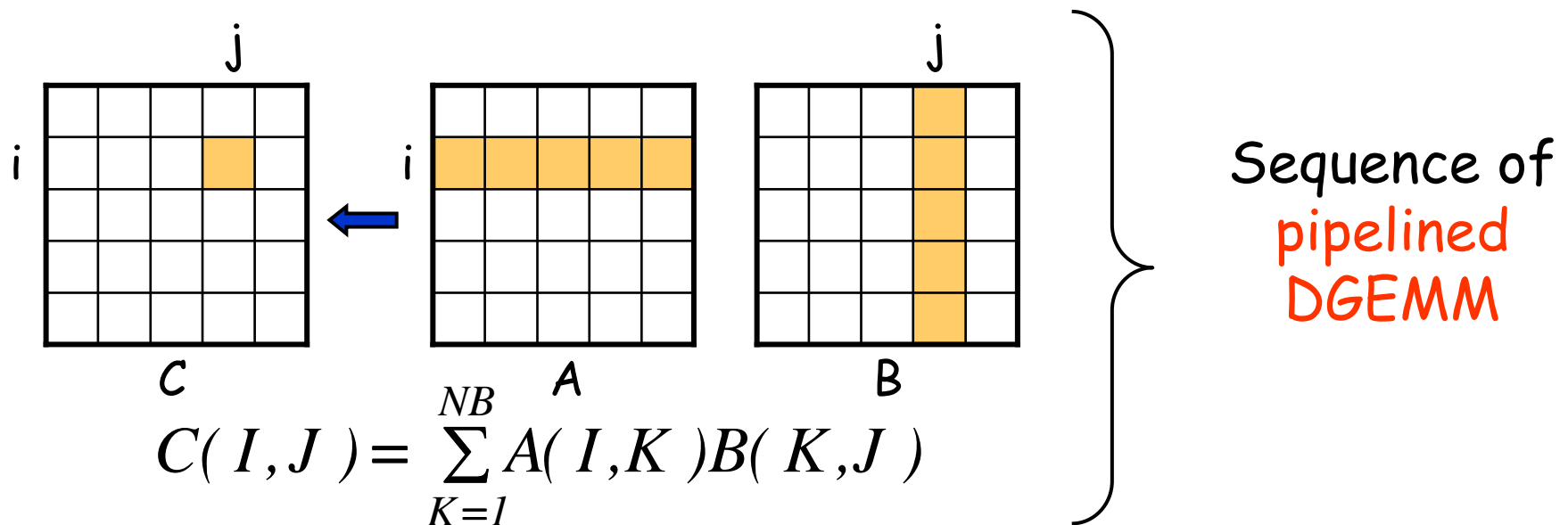
Minor changes in the related code

Similar changes to other DSI functions

# Computational experiments

- **Block matrix multiplication**

- Representative of similar other computations (LU, QR, ...)
- Basic linear algebra **computational kernel**
- Exhibits **sequential dependency** among the tasks
- **Square matrix** of order  $n$  divided in blocks of order  $r$
- $NB = n/r$  is the number of blocks



# Algorithm 1

---

## Client algorithm

```
for I=1, NB (in parallel)
  for J=1, NB (in parallel)
    choose a server
    for K=1, NB
      send C(I,J), A(I,K), B(K,J) to server
      receive C(I,J) from server
    end for
  end for
endfor
```

## Server algorithm

```
receive C(I,J), A(I,K), B(K,J) from client
C(I,J) = C(I,J) + A(I,K) B(K,J)
send C(I,J) to client
```

No cache of  $C(I,J)$  between  
two consecutive values of  $K$

# Algorithm 2

## Client algorithm

```
for I=1, NB (in parallel)
  for J=1, NB (in parallel)
    choose a server
    store C(I,J) in the server
      secondary storage
    for K=1, NB
      send A(I,K), B(K,J) to server
    end for
    retrieve C(I,J) from the server
      secondary storage
  end for
endfor
```

## Server algorithm

```
retrieve C(I,J) from the secondary
storage
```

```
receive A(I,K), B(K,J) from client
C(I,J)=C(I,J)+A(I,K)B(K,J)
```

```
store C(I,J) in the secondary storage
```

Call to the modified DSI  
functions for caching  $C(I,J)$   
between two consecutive  
values of  $K$

# Computational environment

---

- 2.4 GHz PCs connected by a local area network
- 100 Mbits LAN ( shared with other users )
- Parallel ATA disk adapter of 100 MBytes/sec
- Linux 2.4 O.S
- NetSolve 2.0 + DSI + IBP 1.0.4.2

bandwidth  
 $B_r \sim 12.5 \text{ MBytes/sec}$

bandwidth  
 $B_s \sim 100 \text{ MBytes/sec}$



$$B_s / B_r \sim 8$$

(ideal case)

# Computational cost

$T_r$  access time for a real data (64 bit)  
to the remote client

$T_s$  access time for a real data (64 bit)  
to the secondary storage

$$\frac{T_r}{T_s} = \frac{B_s}{B_r} \sim 8$$

Algorithm 1

$$T_1 = 4 NB T_r r^2$$

Algorithm 2

$$T_2 = 2 NB (T_r + T_s) r^2$$

$$T_2 / T_1 = (T_r + T_r/8) / 2T_r \sim 0.56$$

## Execution time (in sec)

	Algorithm 1	Algorithm 2	$T_2/T_1$
n=50 (NB=1)	0.08	0.08	1
n=100 (NB=2)	0.41	0.36	0.87
n=200 (NB=4)	9.5	5.2	0.54
n=300 (NB=6)	14.38	8.5	0.59

Square blocks of order  $r = 50$

	Algorithm 1	Algorithm 2	$T_2/T_1$
n=500 (NB=1)	2.46	1.81	0.73
n=1000 (NB=2)	17.7	12.2	0.68
n=2000 (NB=4)	66.6	44.4	0.66

Square blocks of order  $r = 500$

# Conclusions

---

- Grid and distributed computing **challenges are related** not only to the middleware aspects but also **to computing issues**
- **Caching strategies** are examples of effective methodologies to improve Comm/Comp ratio
- **Need of software tools and environments** able to support such a strategies