An Input/Output LIbrary for cluster of SMP

Adrien Lebre, Yves Denneulin

{*Adrien.Lebre,Yves.Denneulin*}*@imag.fr*

ID-IMAG (UMR 5132) Laboratory, Grenoble, France
BULL - HPC, Échirolles, France.

Plan

Context

Environment

- Cluster of SMPs
- Linux
- High Performance Computing
- Intensive I/O applications
  - CPU bounded application ⇒ I/O bounded application
  - Remote hard drive I/O

Parallel I/O

- Handling concurrent accesses to a same resource (a file)
- Accesses : different in size, in offset

  Example : matrix product

Context

Environment

- Cluster of SMPs
- Linux
- High Performance Computing
- Intensive I/O applications
  - CPU bounded application ⇒ I/O bounded application
  - Remote hard drive I/O

Parallel I/O

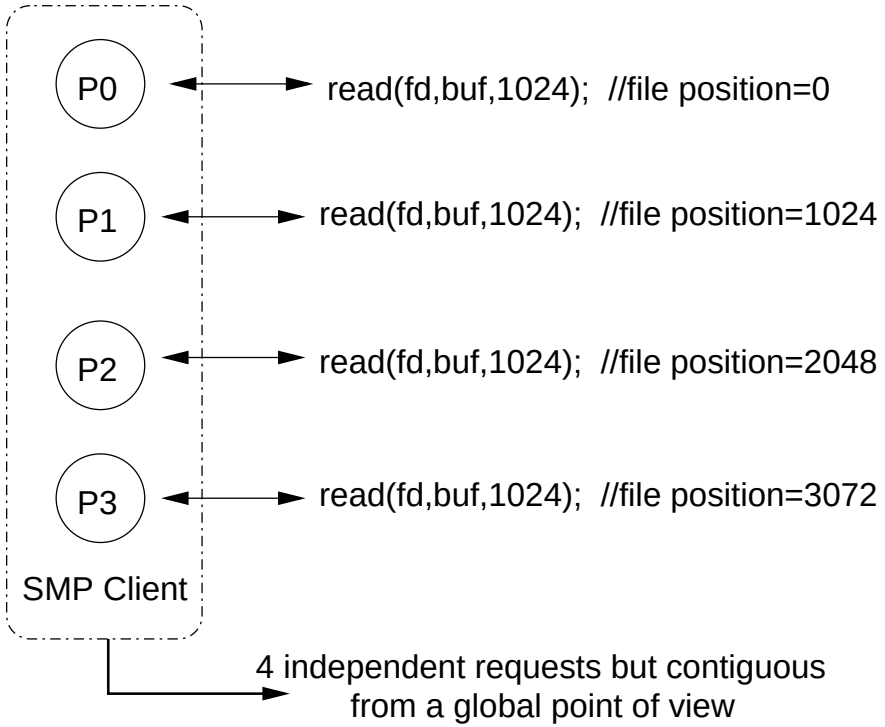
- Handling concurrent accesses to a same resource (a file)
- Accesses : different in size, in offset

  Example : matrix product

Parallel I/O Example

Matrix product

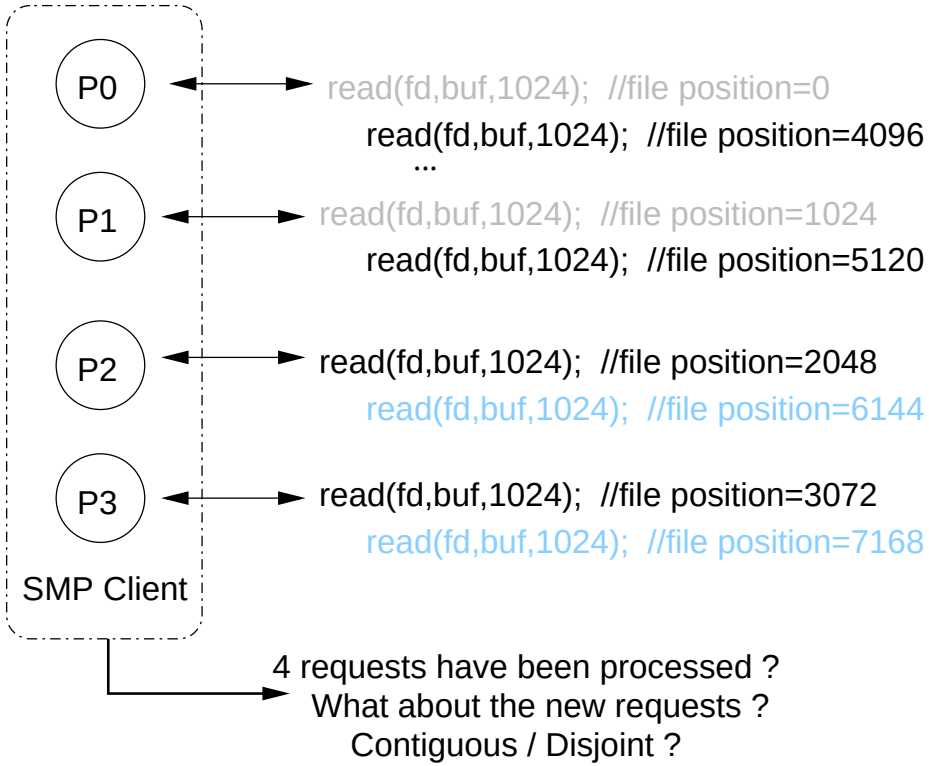
- Specific parts to fetch (according to data distribution: columns, rows, BLOCK/BLOCK, BLOCK/CYCLIC ...)
- Several requests at the same time : disjoint/contiguous
- "lethal" behavior for I/O subsystem



Parallel I/O Example

Matrix product

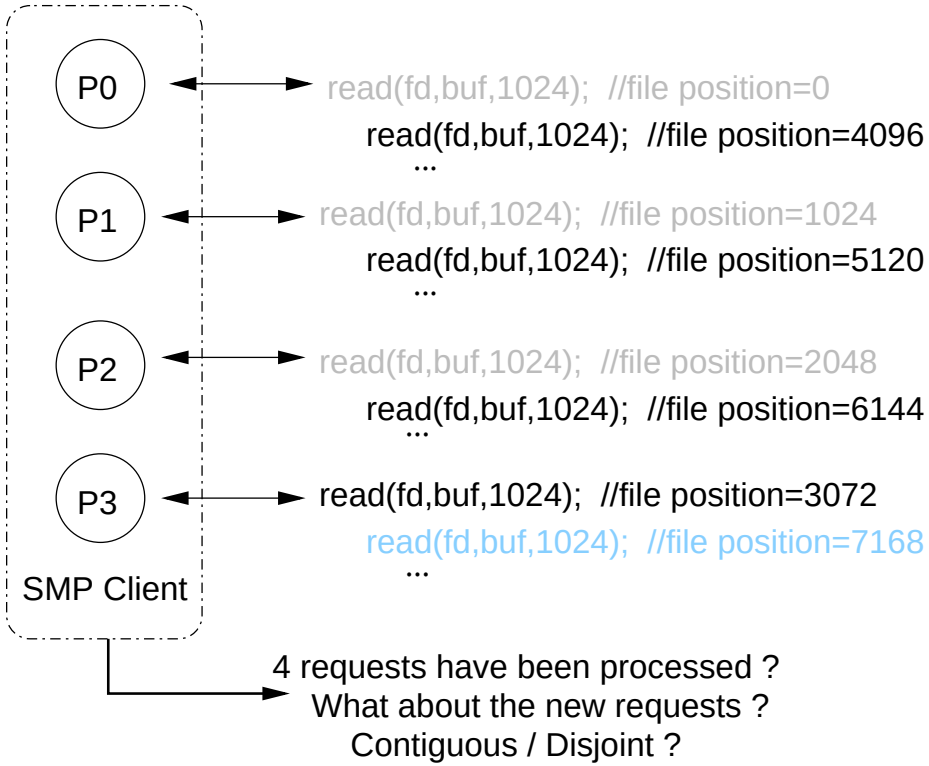
- Specific parts to fetch (according to data distribution: columns, rows, BLOCK/BLOCK, BLOCK/CYCLIC ...)
- Several requests at the same time : disjoint/contiguous
- "lethal" behavior for I/O subsystem



Parallel I/O Example

Matrix product

- Specific parts to fetch (according to data distribution: columns, rows, BLOCK/BLOCK, BLOCK/CYCLIC ...)
- Several requests at the same time : disjoint/contiguous
- "lethal" behavior for I/O subsystem



Parallel I/O

Requirements / constraints

- Methods for disjoint data (readv) ⇒ complexity of API
- Collective operations ⇒ Synchronization mechanisms
- logical view (the files) ⇒ physical placements (block devices)

Available solutions - related works

- Many Parallel File Systems : +/- efficient but hardware dependent
  - "cluster compliant" : PVFS, NFSparallel, GPFS, Lustre
  - Designed for " Parallel I/O" : PIOUS, VESTA ...
- Libraries : Focus on portability aspects
  - A lot ! : MPI I/O is the reference.
- Sophisticated API ⇒ Development overhead / Language bindings

Parallel I/O

Requirements / constraints

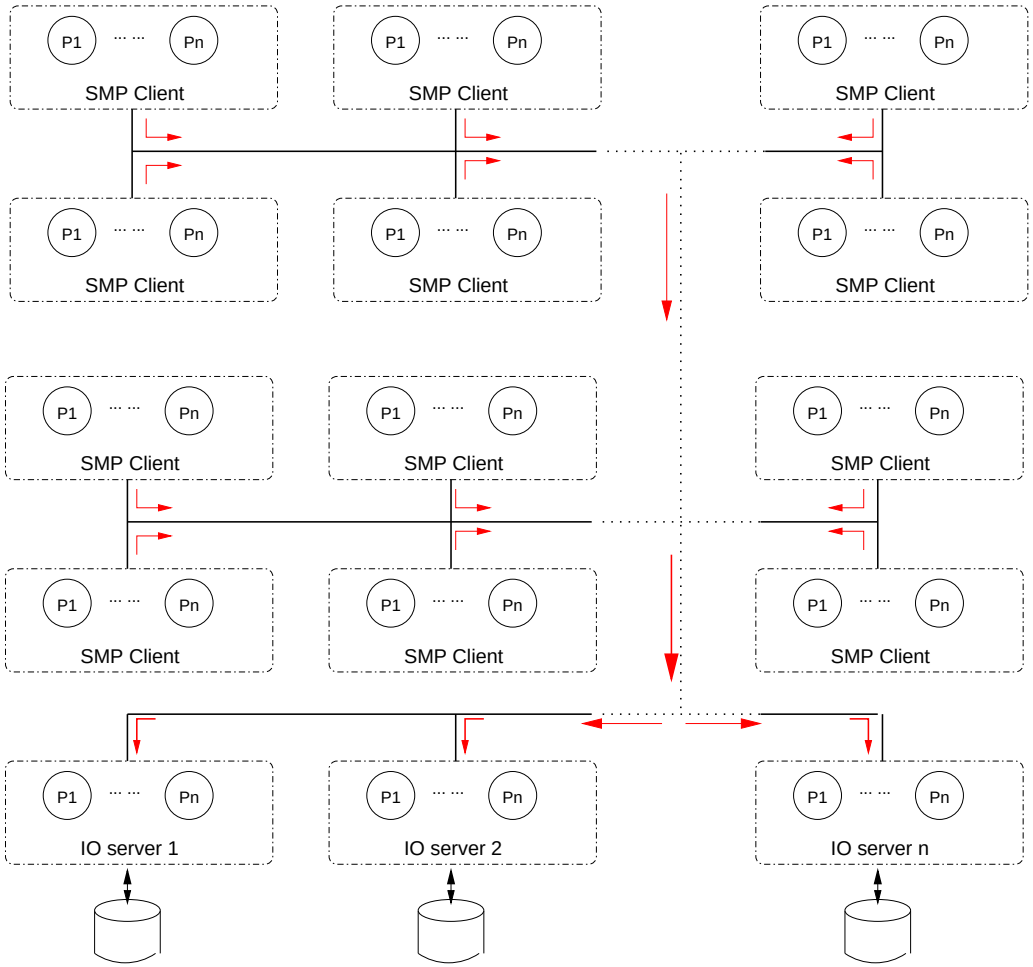
- Methods for disjoint data (readv) ⇒ complexity of API
- Collective operations ⇒ Synchronization mechanisms
- logical view (the files) ⇒ physical placements (block devices)

Available solutions - related works

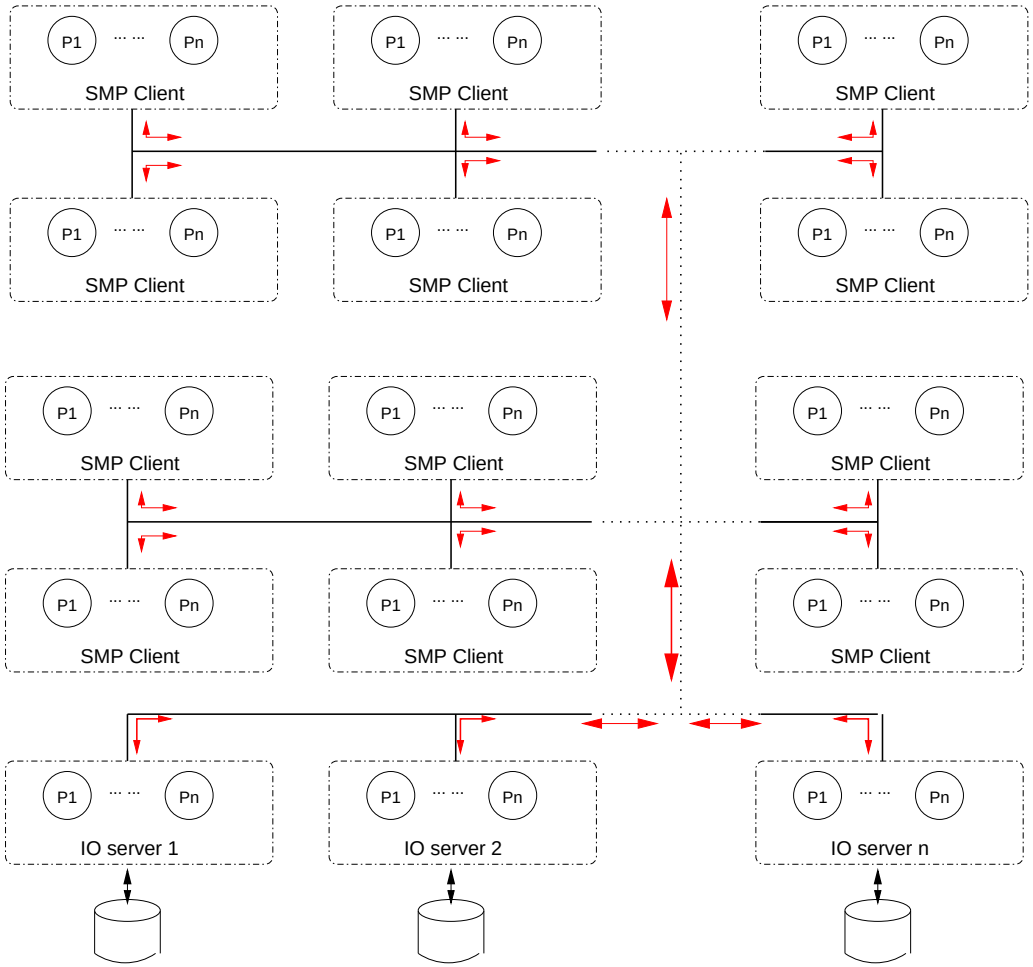
- Many Parallel File Systems : +/- efficient but hardware dependent
  - "cluster compliant" : PVFS, NFSparallel, GPFS, Lustre
  - Designed for " Parallel I/O" : PIOUS, VESTA ...
- Libraries : Focus on portability aspects
  - A lot ! : MPI I/O is the reference.
- Sophisticated API ⇒ Development overhead / Language bindings

Context summary

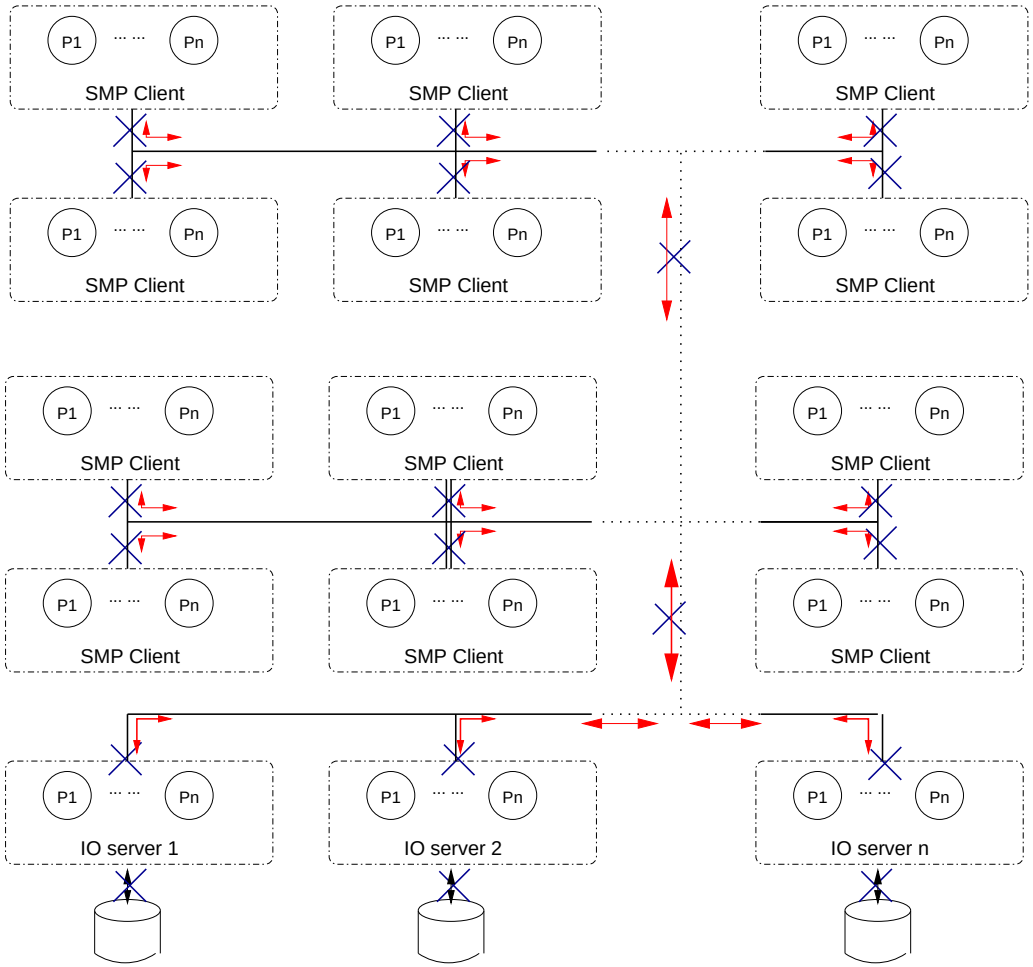
Context summary



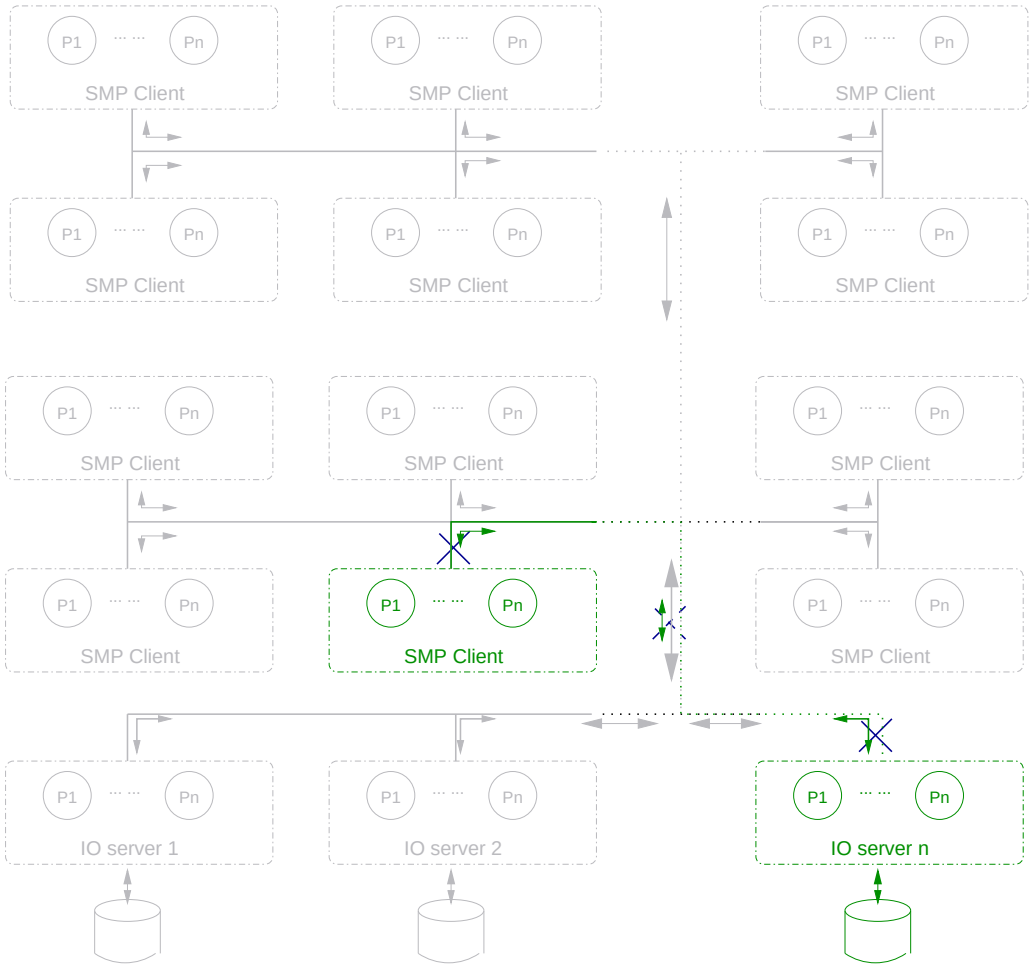
Context summary



Context summary



Context summary



aIOLi system

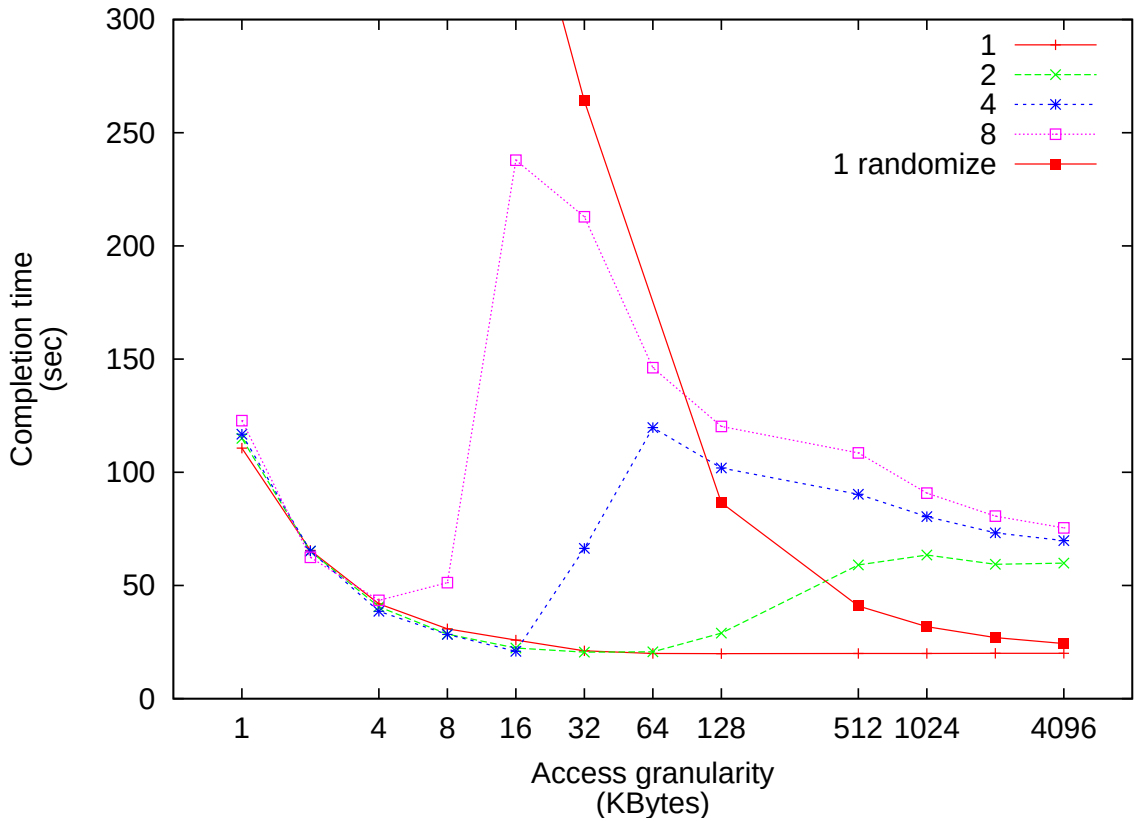
Objectives

- Supply Parallel I/O algorithms
  - scheduling policies
  - aggregating access ⇒ efficiency
  - overlapping access
- Only through the use of the ubiquitous POSIX calls
  - `open/creat/lseek/read/write/close` ⇒ Simplicity
- Minimal overhead
  - avoid expensive synchronisation mechanisms (barrier, ...)

Evaluation of "the Linux" I/O stack

1 GB File decomposition on a SMP

(kernel 2.4.27, IDPOT cluster, NFS version 3, mpich 1.2.5)



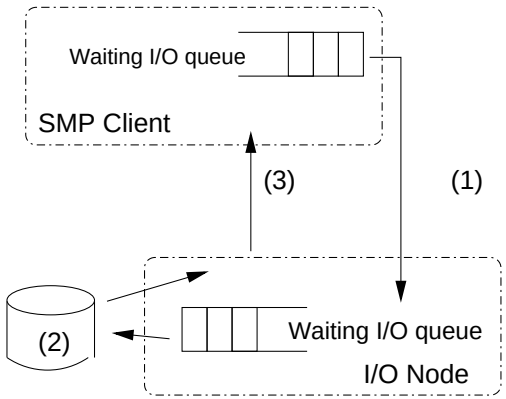
Observations

- 1 process ⇒ Sequential read (optimal)
- + processes ⇒ - performance
- 1 process in random access ⇒ more performance for large access than parallel approach

Fundamental concepts

Define a "think time" window

- Maximize the use of I/O server (bandwidth)
  - At least one request should be in the queue on the server side
- Apply parallel I/O algorithms in the queue on the client side
  - At most one request on the server side !

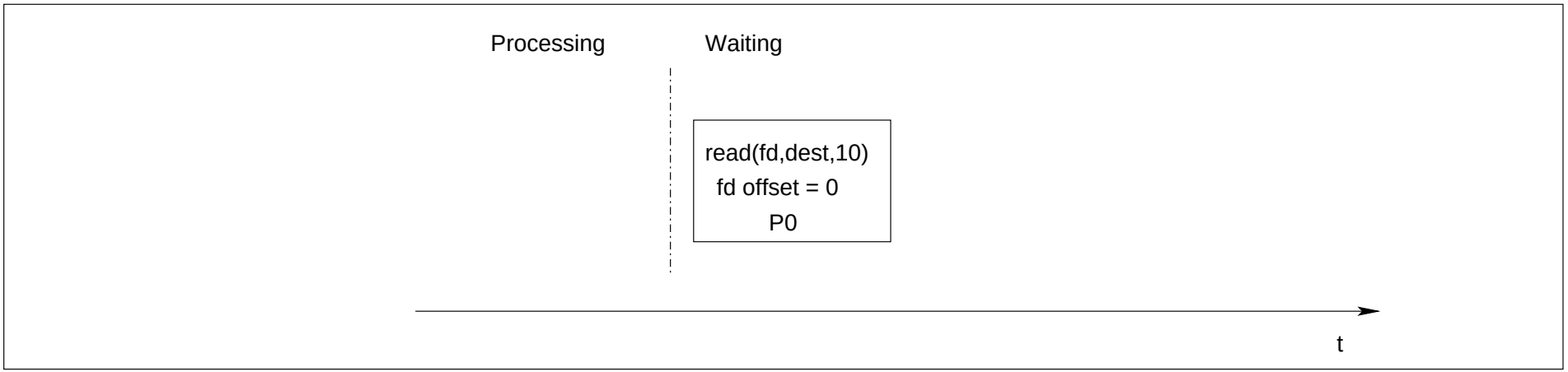


(1) A first request is sent to the file server.
(2) The server processes it on the attached disk.
(3) The reply is returned to the client.

Fundamental concepts

Aggregating example

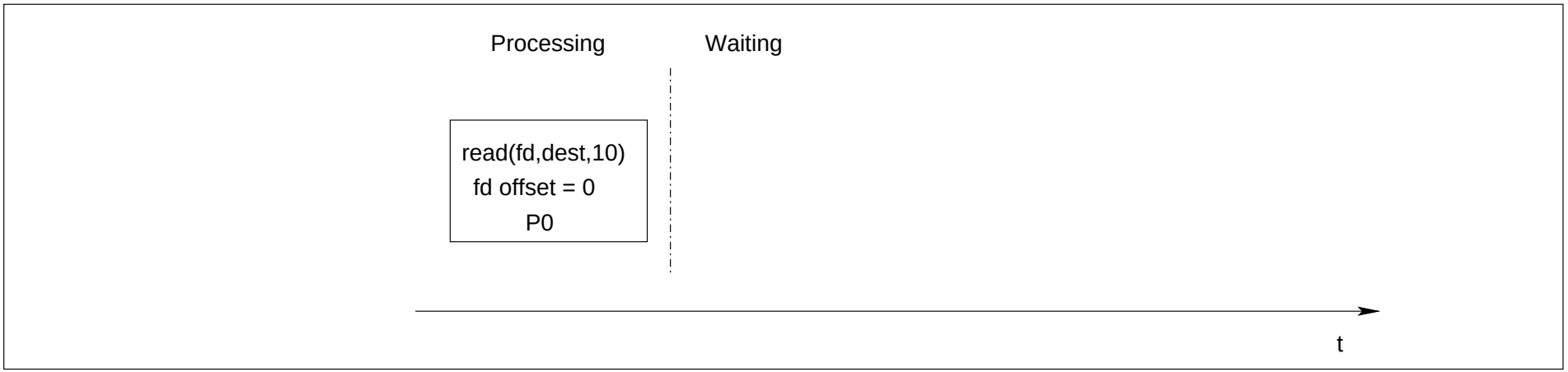
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

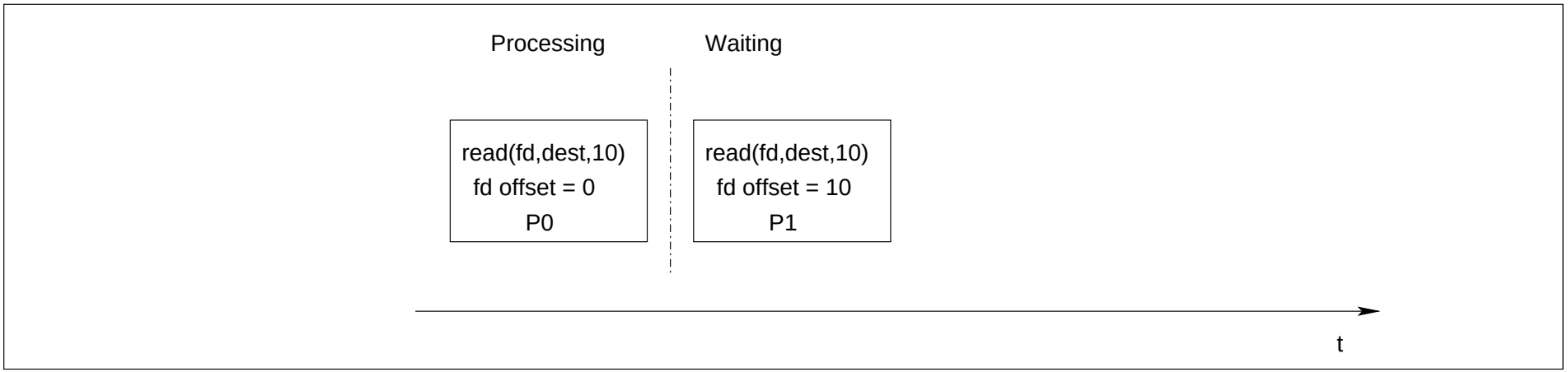
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

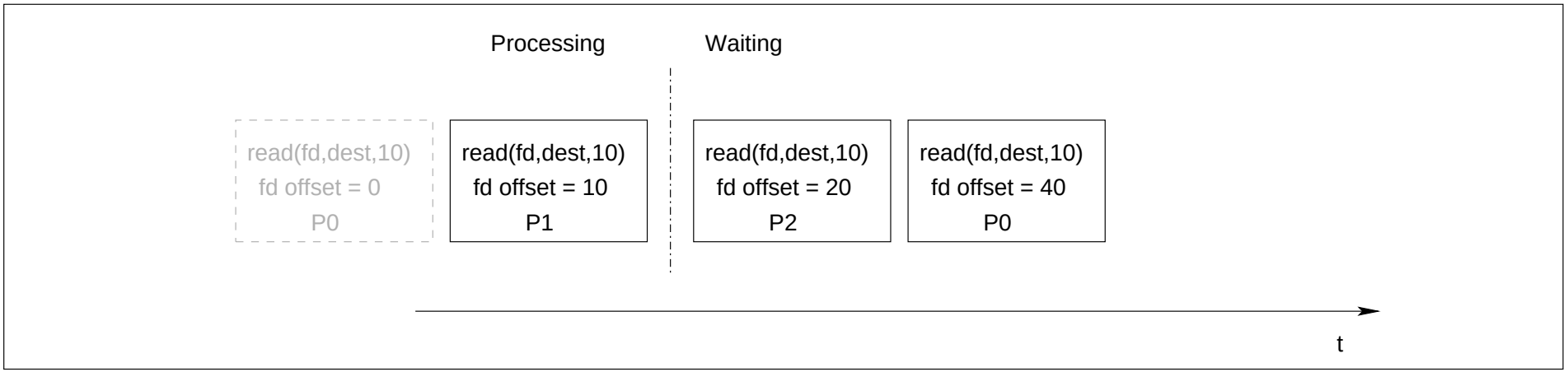
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

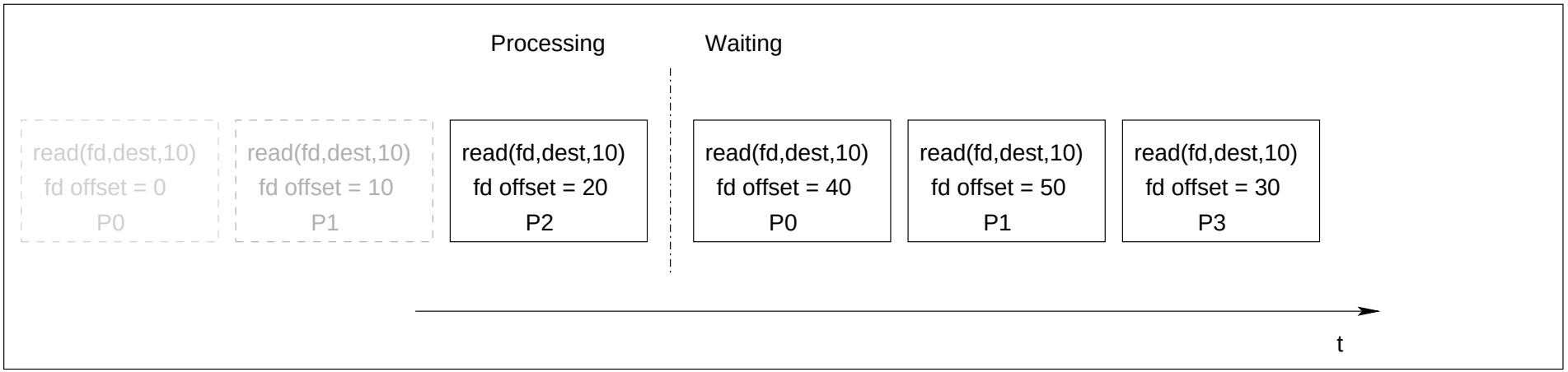
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

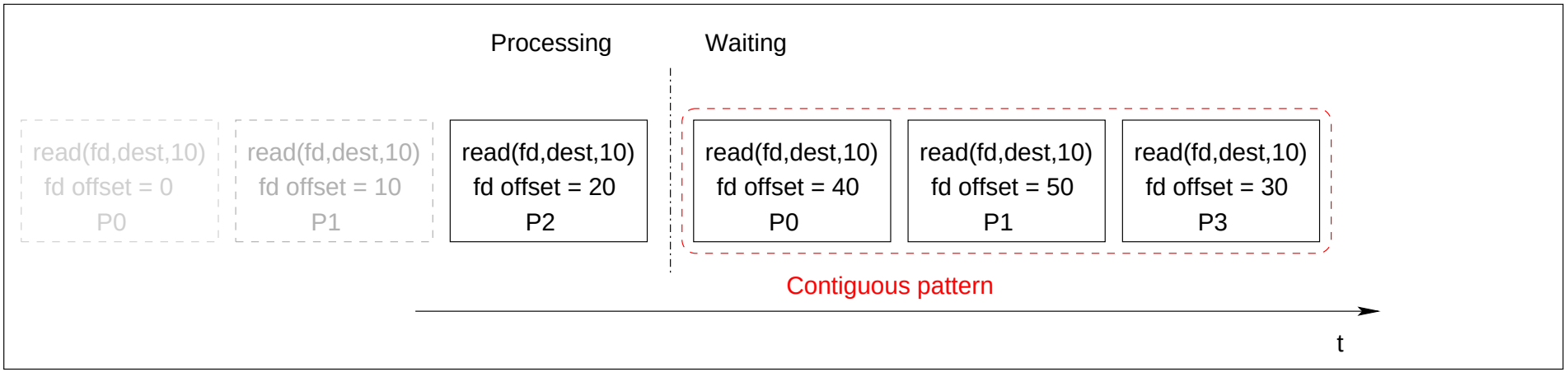
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

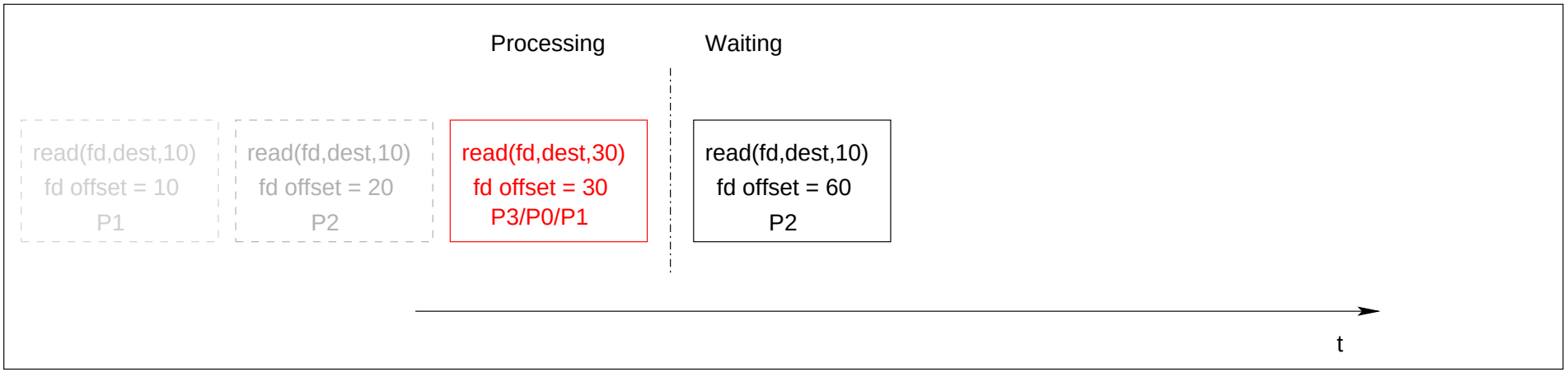
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

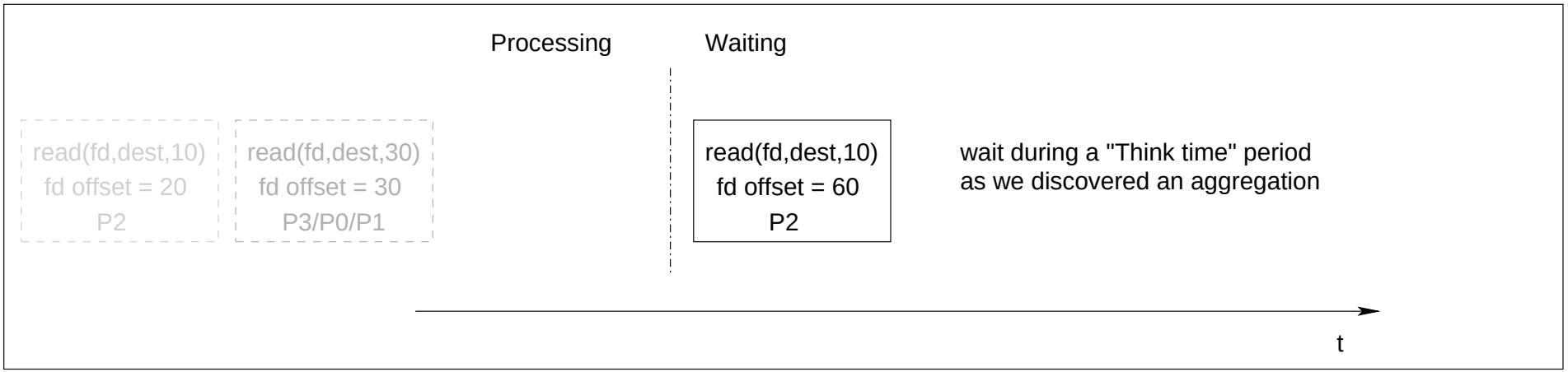
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

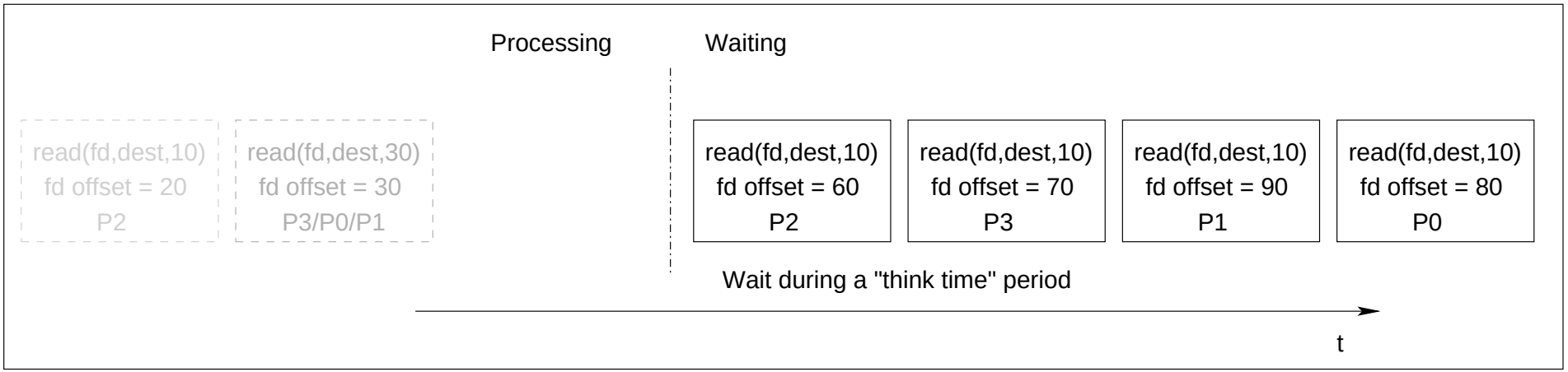
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

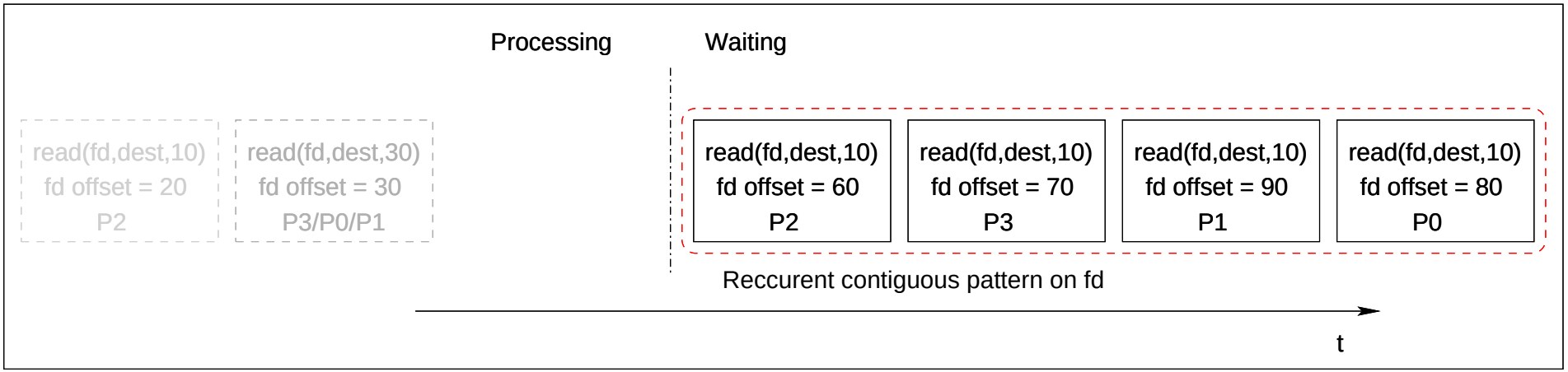
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

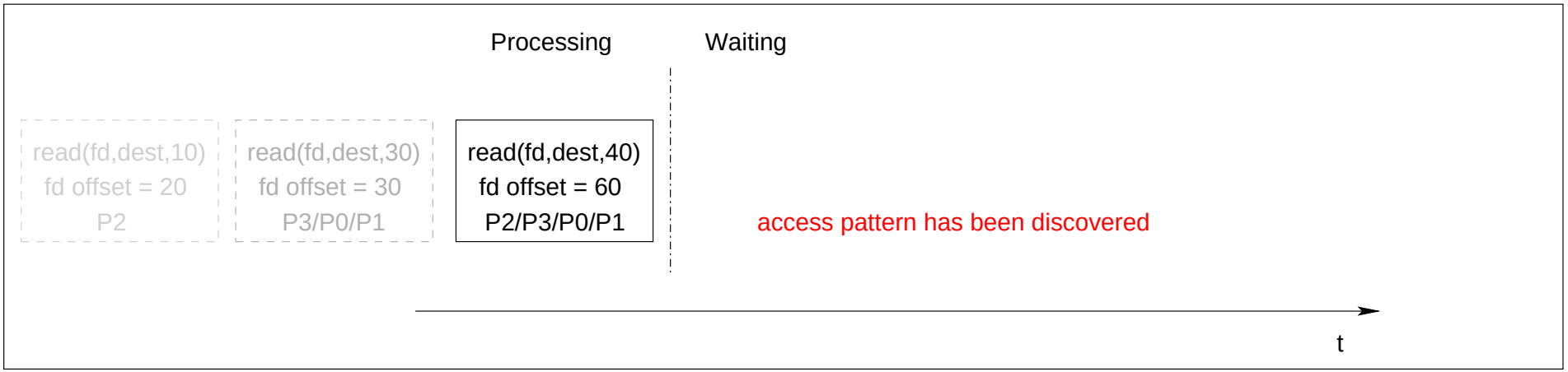
- basic decomposition including 4 processes (granularity=10 bytes)



Fundamental concepts

Aggregating example

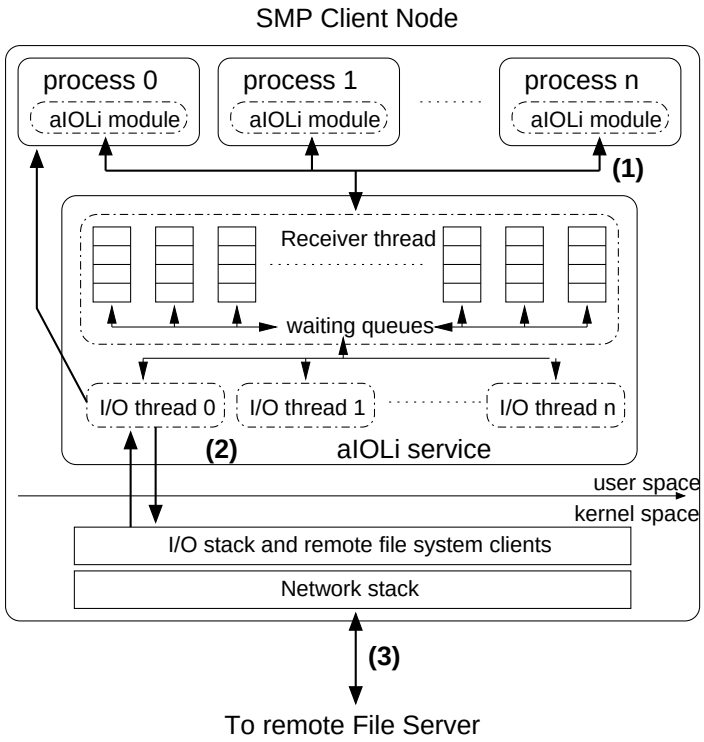
- basic decomposition including 4 processes (granularity=10 bytes)



aIOLi prototype

User library

- A component overloads POSIX calls (linked to the HPC applications)
- aIOLi daemon
  - "Multi-threaded"
  - Includes distinct improvements
  - Processes real I/O calls
- IPC mechanisms and shared memory

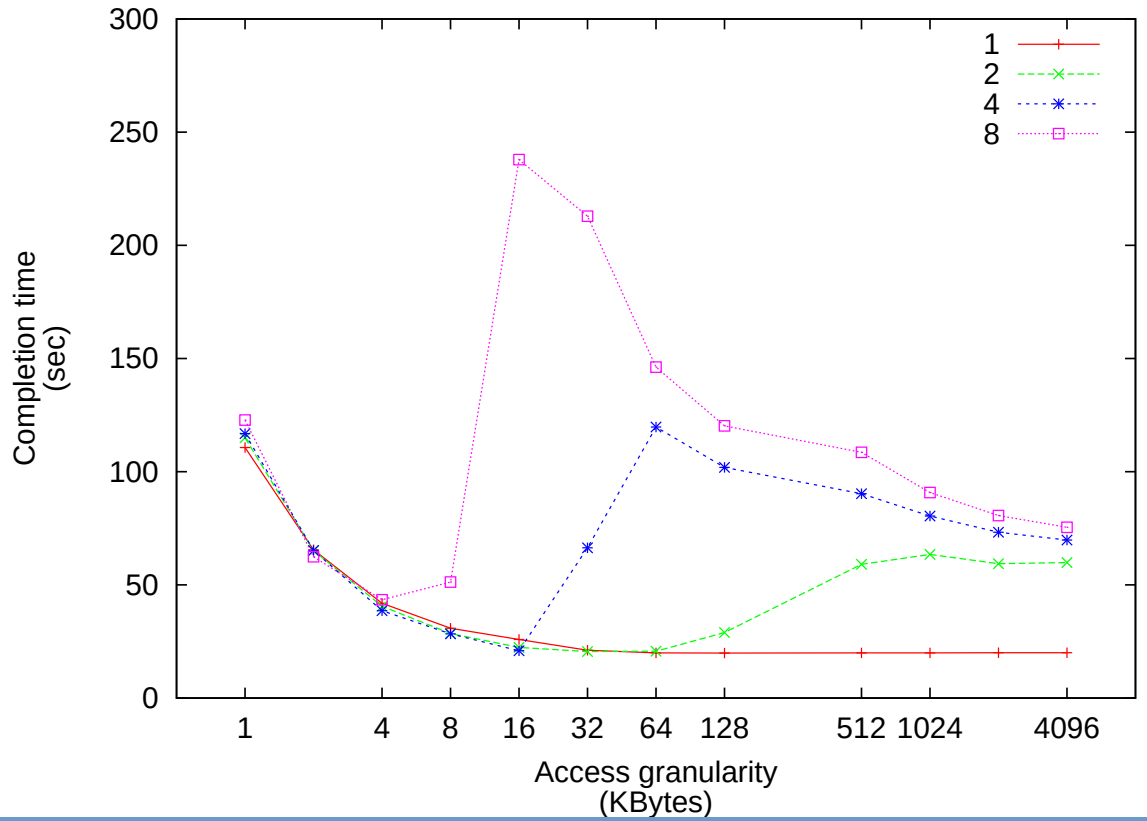


Evaluation : POSIX vs aIOLi

1 GB File decomposition on a SMP

(kernel 2.4.27, IDPOT cluster, NFS version 3, mpich 1.2.5)

compiled without aIOLi

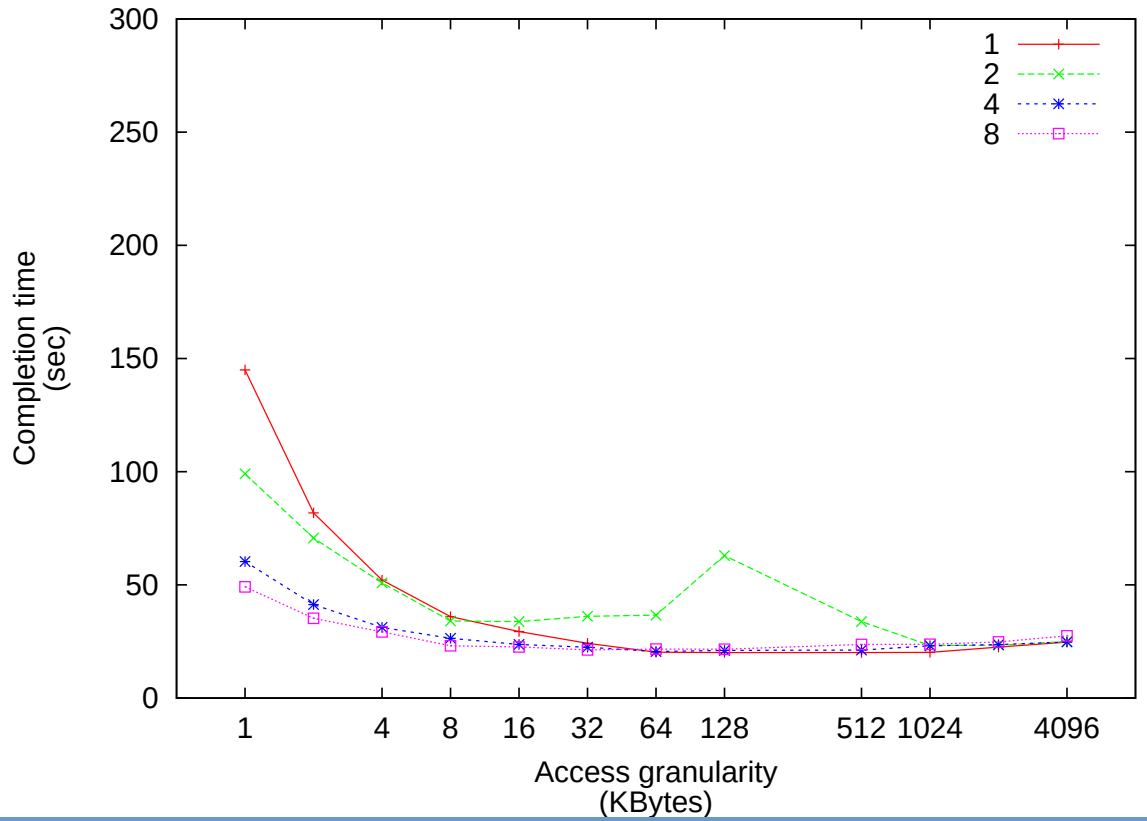


Evaluation : POSIX vs aIOLi

1 GB File decomposition on a SMP

(kernel 2.4.27, IDPOT cluster, NFS version 3, mpich 1.2.5)

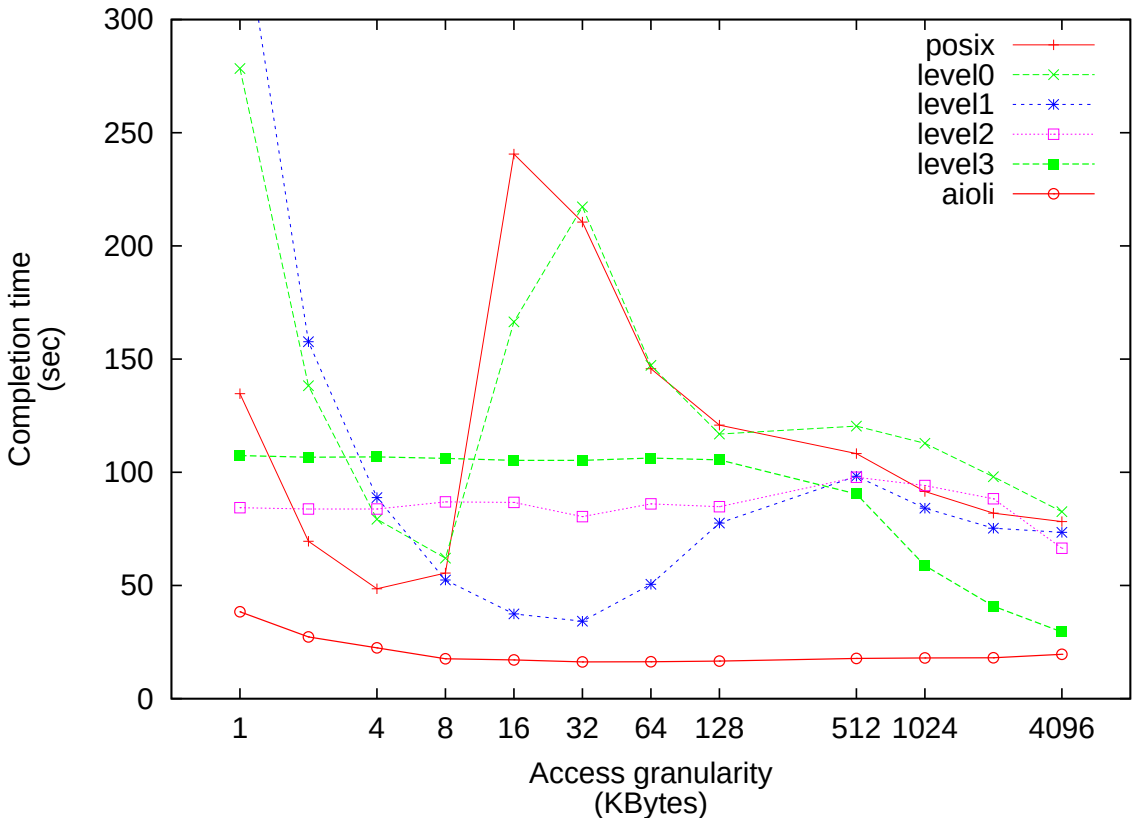
compiled with aIOLi



Evaluation : MPI I/O vs aIOLi

1 GB File decomposition including 8 MPI instances on a SMP

(kernel 2.4.27, IDPOT cluster, NFS version 3, mpich 1.2.5, ROMIO)



Observations

- MPI I/O : explicit access pattern
- For all levels, aIOLi provided the best results

Conclusion

Positive results

- Efficient
- Simplicity of the API ⇒ POSIX
- No requirements for inter-process synchronization.

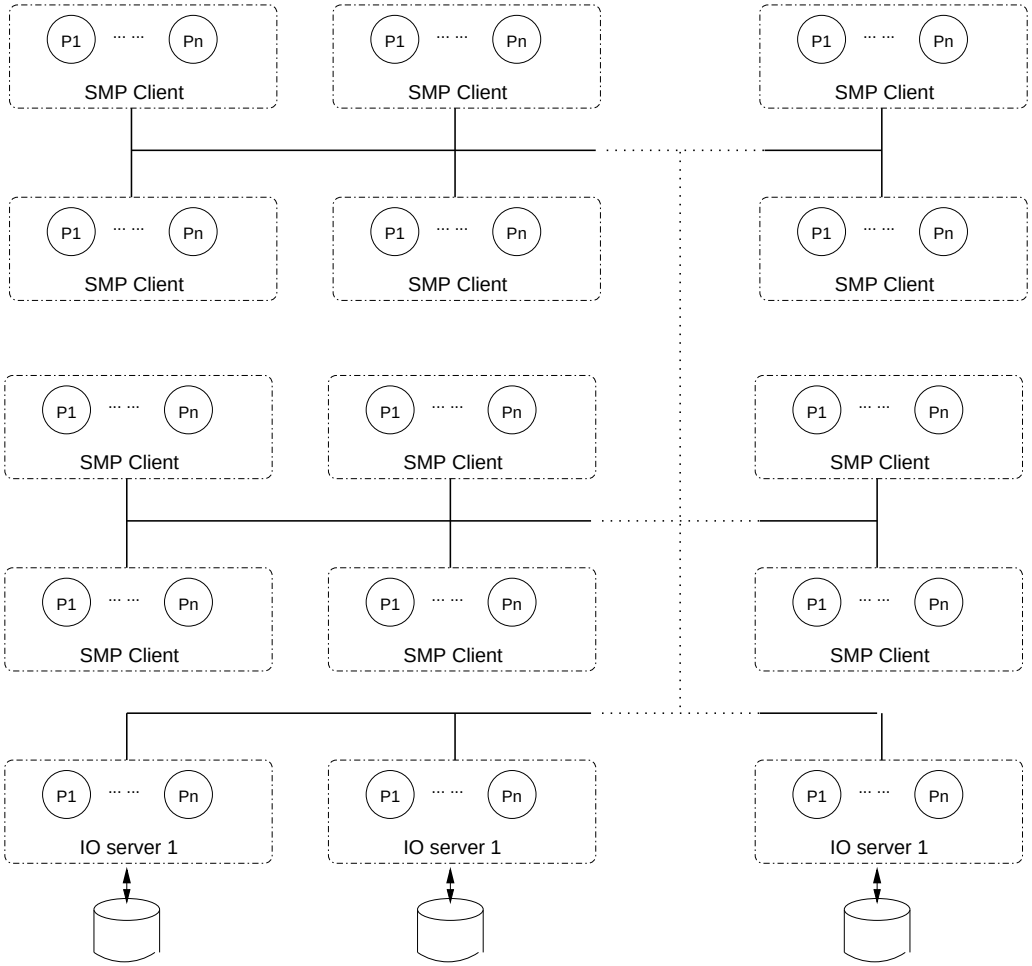
Current constraints

- Centralized distributed file system (such as NFS) vs Parallel FS
- Reduce overhead for single access
- Kernel scheduler dependent

Current and future works

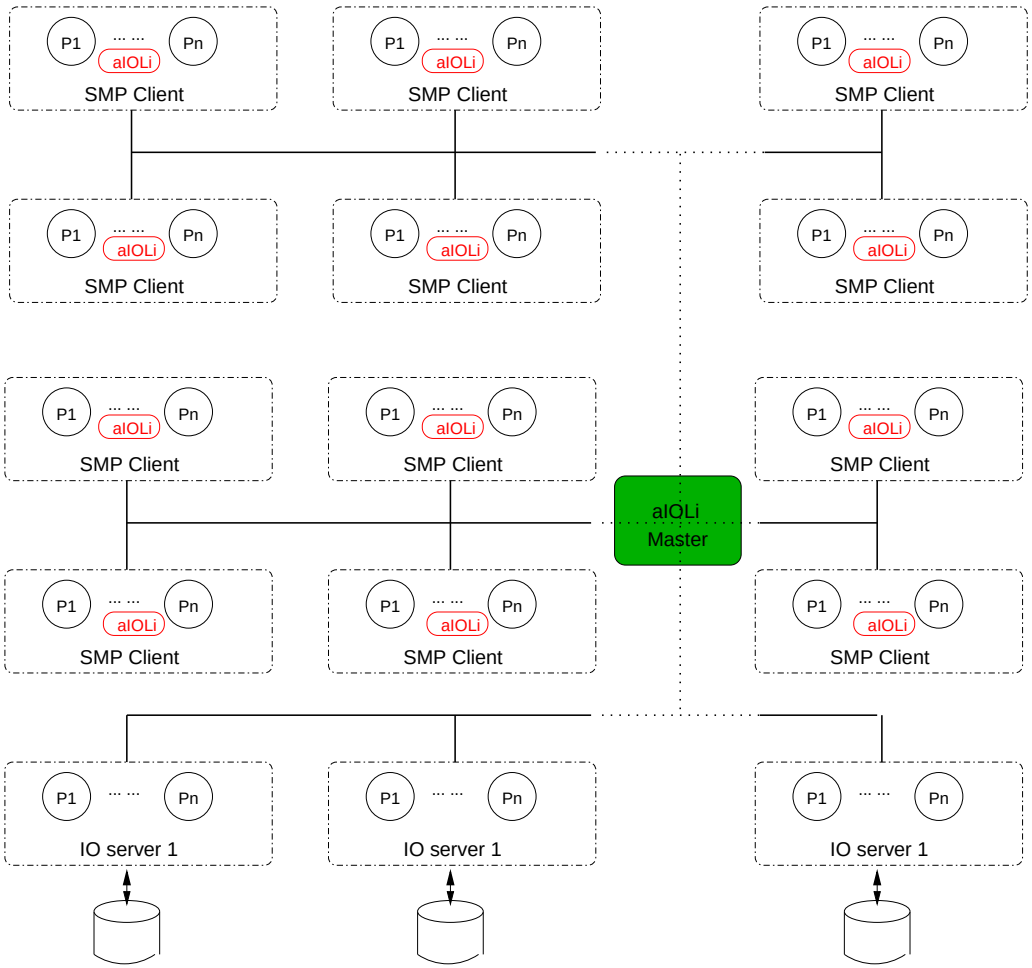
- Add Data striping considerations (stabilization phase)
- Implement a patch for the VFS (summer 2005)
- Evaluation on bigger SMP and Lustre
- Coordination between several SMPs, in progress (Master)
- The grid ...

Conclusion - summary

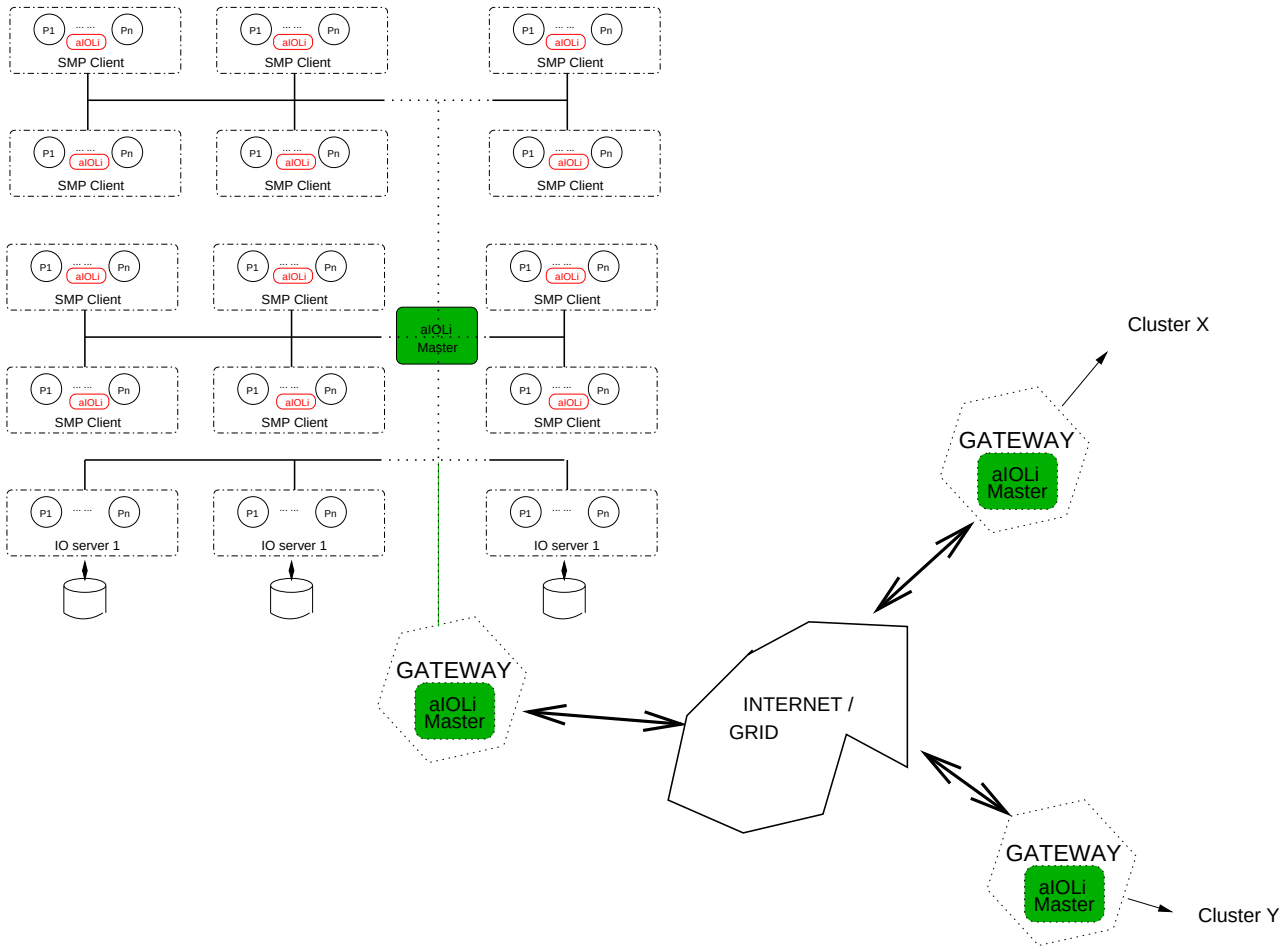


Conclusion - summary

Conclusion - summary



Conclusion - summary



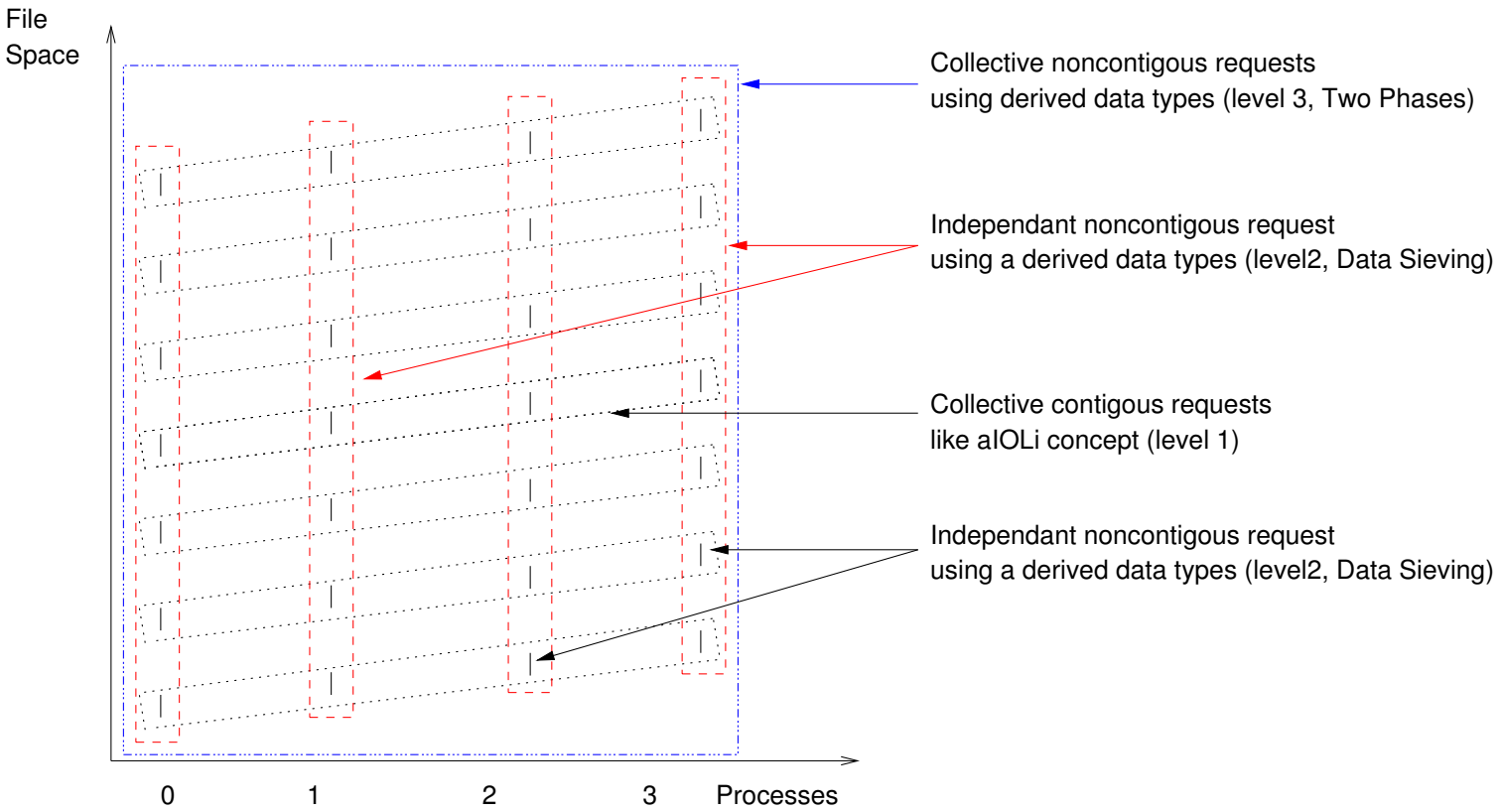
Questions ?

http://aioli.imag.fr

LIPS Project
BULL - INRIA - ID Laboratory

Thanks

MPI I/O improvements



MPI I/O – Four levels
Representing increasing amounts of data per request
[Thakhur/Gropp/Lusk02]