Autonomic Solutions for Parallel and Distributed Data Stream Processing (Auto-DaSP) - Euro-Par 2017 Workshop

STREAMING IN THE PGAS ERA

Marco Aldinucci, Maurizio Drocco
University of Torino, Italy

UNIVERSITÀ DEGLI STUDI DI TORINO

Outline

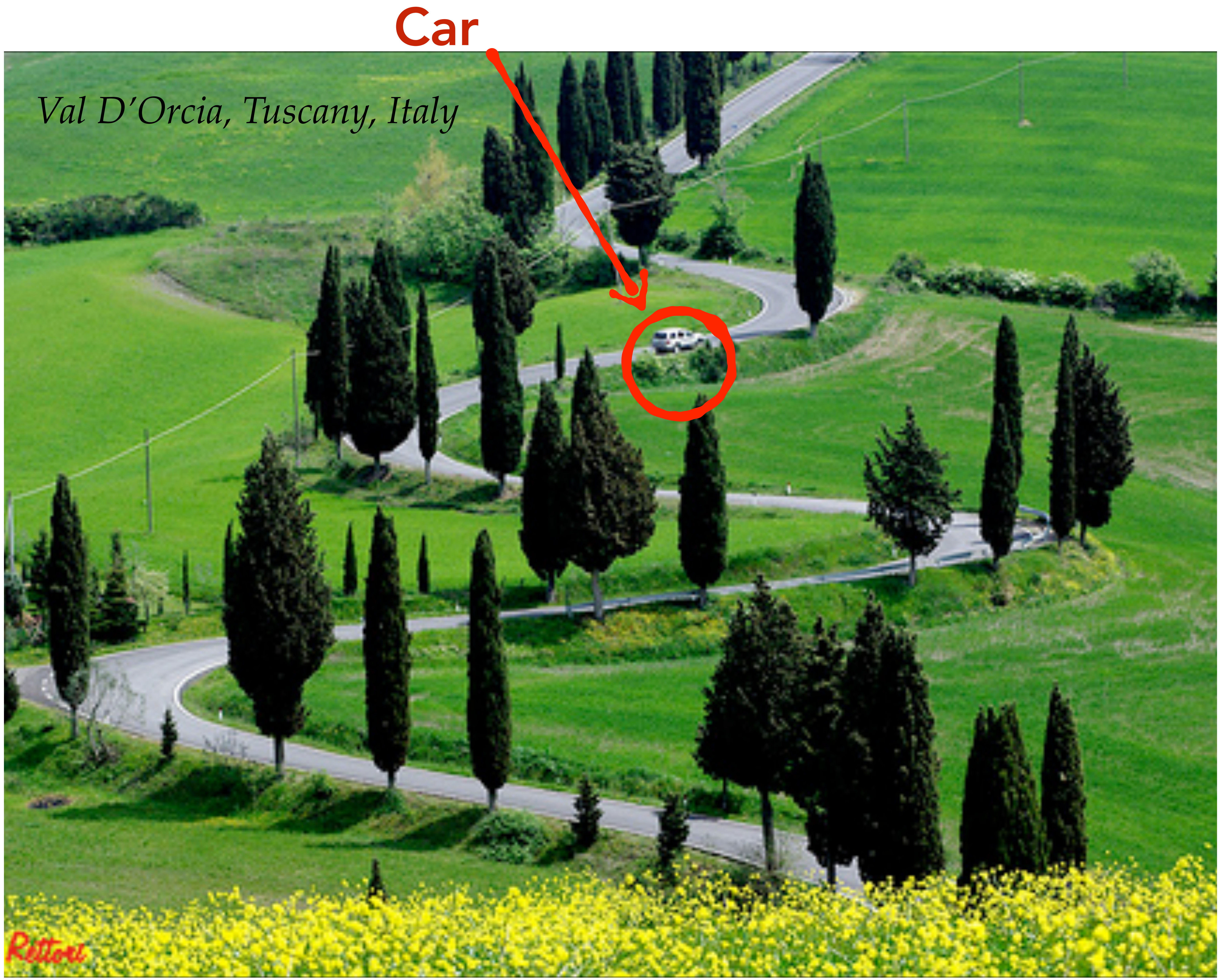
- Programming models
- Distributed Memories: the PGAS model
  - Some example
- Streaming
  - Some example
- Streaming with PGAS

Low-level parallel programming models

- Message-Passing
  - Scalability and performance
  - Developer-based precise knowledge of code and overhead
  - Processes + communications (symmetric/collective, blocking/nonblocking)
- Shared-Memory
  - Productivity
  - Global and uniform vision of data layout
  - Threads + synchronisations mechanisms (mutex, atomics, transactions, …)

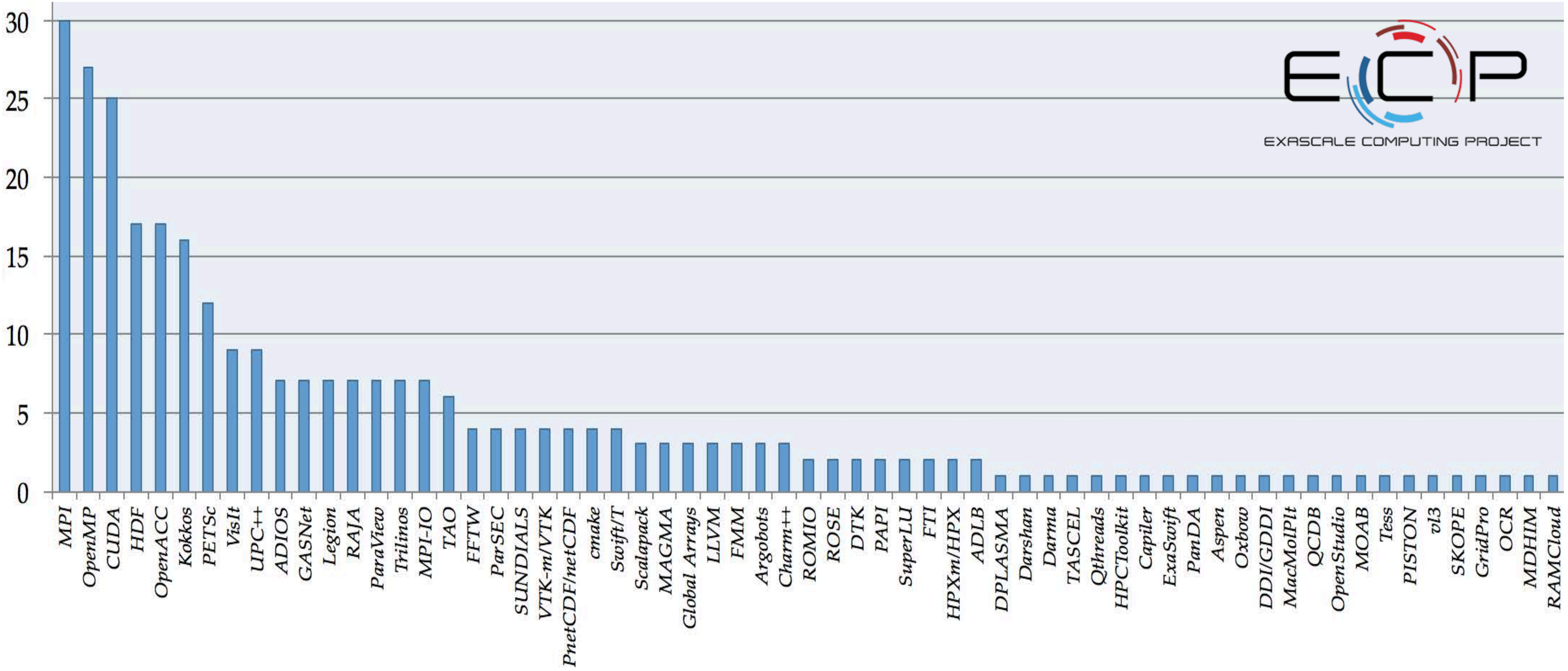
MPI is like a car, you can drive data where you like

–D.K. Panda, leader MVAPICH project at Ohio state Uni.



EXPECTATIONS

REALITY



Anyway, MPI is no. 1 in HPC

Courtesy of P. Messina, director of ECP.
No. of software proposals in US ECP 2017

DISTRIBUTED SHARED (VIRTUAL) MEMORY — DSM OR DVSM

- Physically separated memories can be addressed as one logically shared address space
- Hardware or software. Conceptually similar to Virtual Memory
- Designed to distributed platform transparent to programmer i.e. **"simplify programming"**
- "Vanilla" API
  - **`read(addr)`**
  - **`write(value, adds)`**
  - **`lock/unlock`**

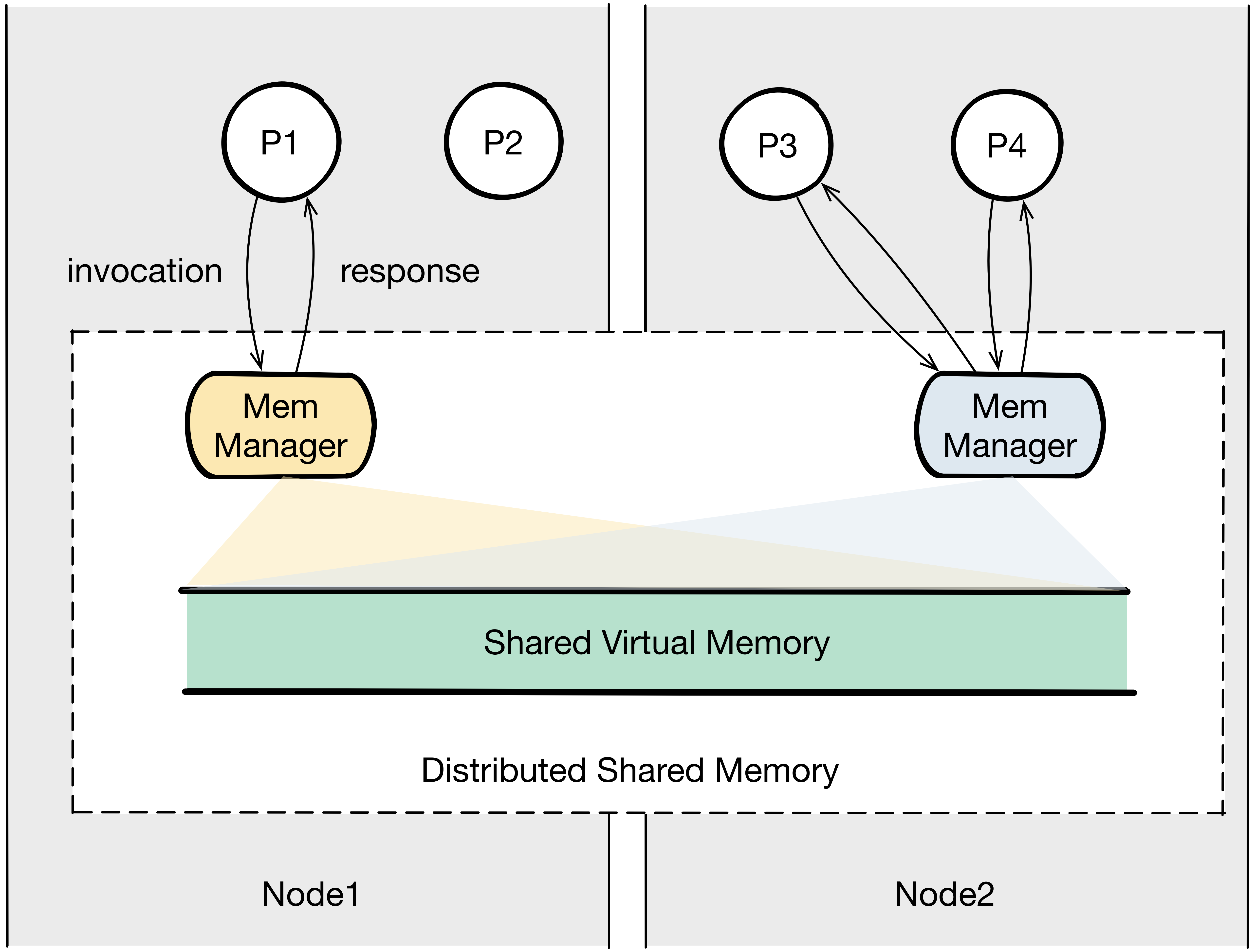


Table A. Software DSM implementations.

Implementation	Type of implementation	Type of algorithm	Consistency model	Granularity unit	Coherence policy
IVY	User-level library + OS modification	MRSW	Sequential	1 Kbyte	Invalidate
Mermaid	User-level library + OS modifications	MRSW	Sequential	1 Kbyte, 8 Kbytes	Invalidate
Munin	Runtime system + linker + library + preprocessor + OS modifications	Type-specific (SRSW, MRSW MRMW)	Release	Variable size objects	Type-specific (delayed update, invalidate)
Midway	Runtime system + compiler	MRMW	Entry, release, processor	4 Kbytes	Update
TreadMarks	User-level	MRMW	Lazy release	4 Kbytes	Update, invalidate
Blizzard	User-level + OS kernel modification	MRSW	Sequential	32-128 bytes	Invalidate
Mirage	OS kernel	MRSW	Sequential	512 bytes	Invalidate
Clouds	OS, out of kernel	MRSW	Inconsistent, sequential	8 Kbytes	Discard segment when unlocked
Linda	Language	MRSW	Sequential	Variable (tuple size)	Implementation-dependent
Orca	Language	MRSW	Synchronization dependent	Shared data object size	Update

Already mature 20 years ago,
now quite rotten

P. Jelica, M. Tomasevic, and V. Milutinovic. "Distributed shared memory: Concepts and systems." IEEE Parallel & Distributed Technology: Systems & Applications 4.2 (1996): 63-71.

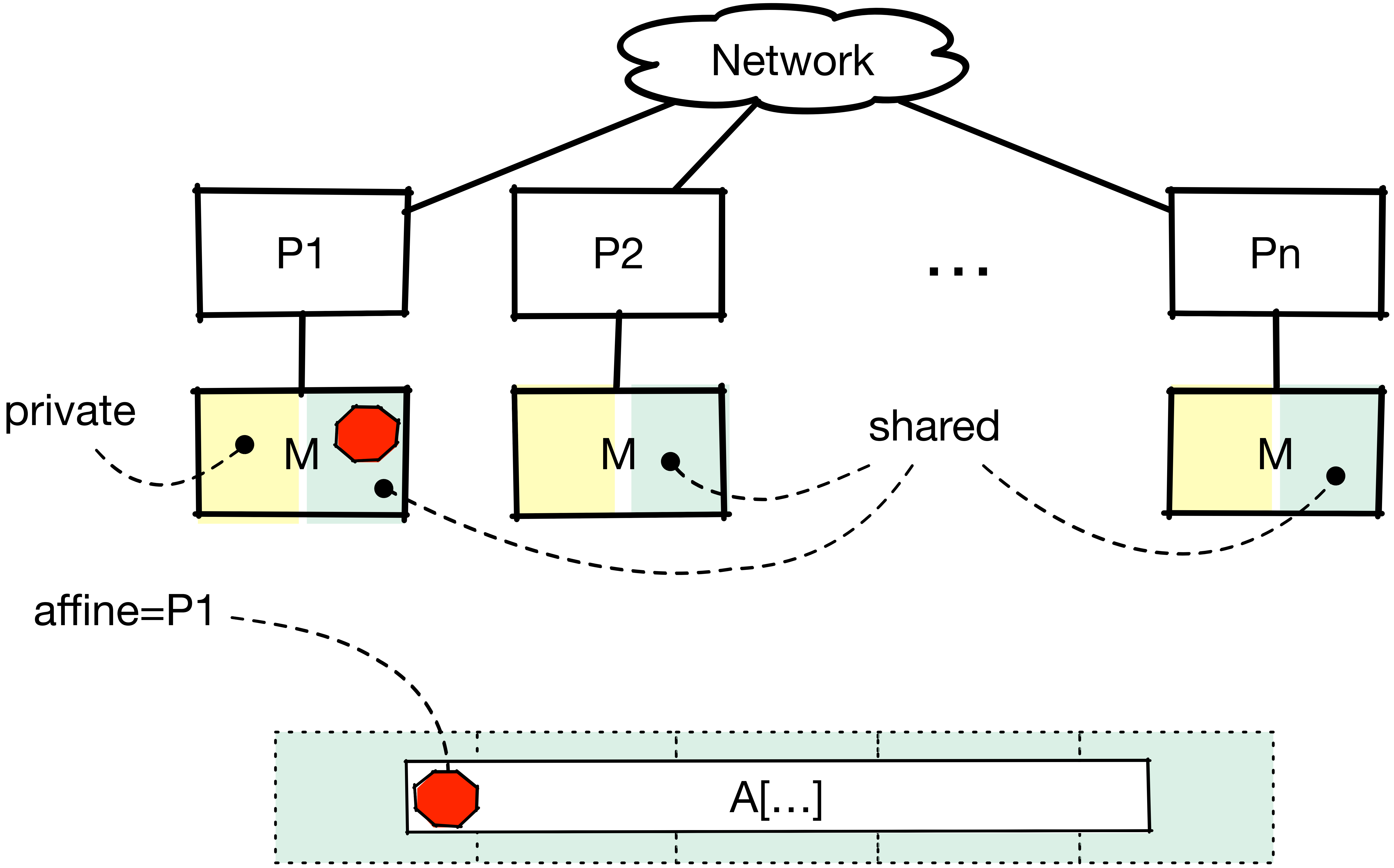
DSM: Why they failed

- Started to simplify distributed code
- To make them efficient, we made the memory consistency model very complex
- This seriously affect the coding effort

PGAS PROGRAMMING MODEL (DSM EVOLVED)

- A set of processor, each with own local memory
- Part managed as private, part as shared
  - Sharing implemented HW or SW
- Explicitly NUMA
  - Each location has an affinity with a processor
  - Model differentiates between local and remote data partitions
- Explicitly partitioned
  - Collective synchronisations, i.e. barriers and fences

PGAS System at Bare Bones



Partitioned Global Address Space (PGAS)

Approaches

Languages

- Unified Parallel C (UPC)
- Co-Array Fortran (CAF)
- X10
- **Chapel**
- STAPL
- Titanium

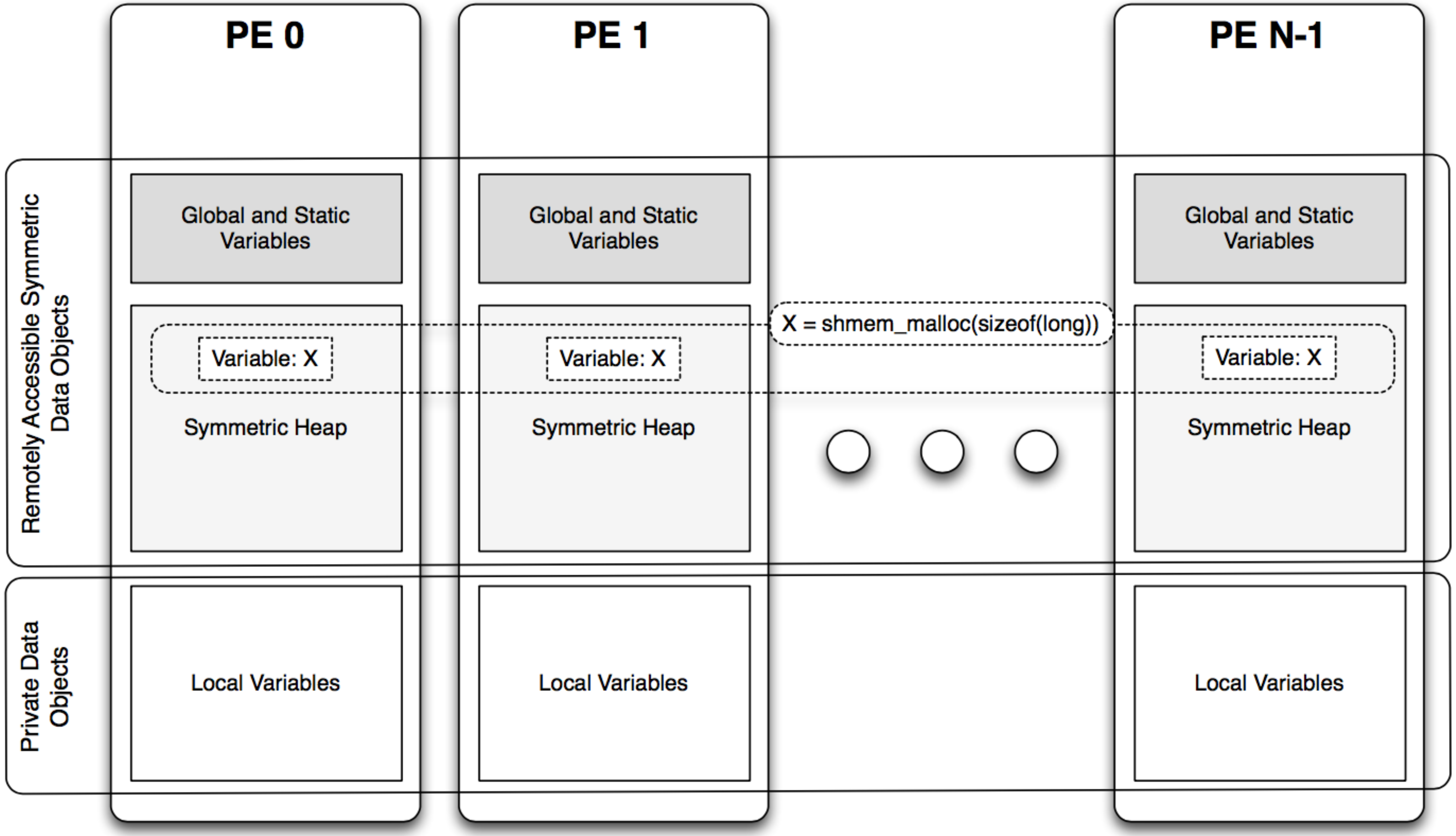
Libraries

- UPC++
- **OpenSHMEM**
- Global Arrays
- **DASH**
- …

```
#include <libdash.h>

int main(int argc, char* argv[]) {
  dash::init(&argc, &argv);

  int myid = dash::myid();
  int size = dash::size();

  dash::array<int> key(size);

  if(myid==0) {
    for(i=0; i<size; i++) key[i]=compute_key(...);
  }

  dash::barrier();

  cout<<"Hello␣from␣unit␣"<<myid<<"␣of␣"
      <<size<<"␣my␣key␣is"<<key[myid]<<endl;

  dash::finalize();
}
```

DASH Hello World

K. Furlinger, C. Glass, J. Gracia, et al. DASH: Data Structures and Algorithms with Support for Hierarchical Locality. In Euro-Par Workshops, 2014.



OpenSHMEM "symmetric" memory model

formerly know as Cray SHMEM (1993)

OpenSHMEM
A old fashioned PGAS

- Perform computations in separate address spaces and explicitly pass data to and from different processes in the program

- API
  - Library Setup: init, query
  - **Symmetric** Data Object Management: allocation, deallocation, reallocation:
  - Remote Memory Access: **get** (private←shared), **put** (shared←private)
  - **Atomics**: swap, inc, add, was, fetch_add
  - **Synchronisation** and Ordering: **fence** (i.e. flush inflight ops), quiet (ensure other completion), **barrier**
  - Collective Communication: **broadcast**, **collection** (i.e. gather), **reduction**
  - Mutual Exclusion: **lock**, **testlock**, **unlock**

OpenSHMEM hello world!

```
#include <stdio.h>
#include <mpp/shmem.h>
int main (int argc, char **argv){

int me, npes;

start_pes (0); /*Library Initialization*/
me = _my_pe ();
npes = _num_pes ();
printf ("Hello World from node %d of %d\n", me, npes);
return 0;
}
```

OpenSHMEM Example

```c
#include <stdio.h>
#include <shmem.h>

long pSync[_SHMEM_BARRIER_SYNC_SIZE];
int x = 10101;

int main(void)
{
   int i, me, npes;

   for (i = 0; i < _SHMEM_BARRIER_SYNC_SIZE; i += 1){
      pSync[i] = _SHMEM_SYNC_VALUE;
   }

   shmem_init();
   me = shmem_my_pe();
   npes = shmem_n_pes();

   if(me % 2 == 0){
      x = 1000 + me;
      /*put to next even PE in a circular fashion*/
      shmem_int_p(&x, 4, (me+2)%npes);
      /*synchronize all even pes*/
      shmem_barrier(0, 1, (npes/2 + npes%2), pSync);
   }
   printf("%d: x = %d\n", me, x);
   return 0;
}
```

CHAPEL LIST WALK

```
class Node {
  var data: real;
  var next: Node;
}

// List init (seq, with remote operations)
var head    = new Node(0);
var current = head;
for i in 1..numLocales-1 do
  on Locales[i] {
    current.next = new Node(i);
    current      = current.next;
  }

// List walk (seq)

current = head;
while current {
  writeln("node with data = ", current.data, " on locale
", current.locale.id);
  current = current.next;
}
writeln();
```

```
// Data-driven List walk (work done in parallel
// Each locale WRITE its own data (own-compute)

current = head;
while current {
  on current {
    writeln("node with data = ", current.data, " on locale
", here.id);
    current = current.next;
  }

// Deallocate (seq)

current = head;
while current {
  on current {
    var ptr = current;
    current = current.next;
    delete ptr;
  }
}
}
```

PGAS

- Overcomes some DSM criticality
- Aiming at productivity for large scale
  - To overcome MPI —- not succeeding yet
  - Most of them build on top or interoperate with MPI
- Designed for weak scalability (own-compute) → Data Parallelism
  - Not for streaming

COROUTINES: SEMANTICS IN SEARCH OF A SYNTAX

by M. Douglas McIlroy

Oxford University and

Bell Telephone Laboratories, Incorporated.

ABSTRACT: Unlike subroutines, coroutines may be connected, and reconnected, in nonhierarchical arrangements. Coroutines are particularly useful for generating and processing data streams. Semantics for coroutines are developed and examples are given.

Streams!

Coroutines: McIlroy, 1968

Symposium on Theoretical Programming Novosibirsk – 1972


Gita
Arvind
Dennis
Data Flow Workshop MIT Endicott House – 1977

IEEE STC Dataflow and Beyond

HTTP://DFSTC.CAPSL.UDEL.EDU

IEEE STC
Parallel Model & system
DataFlow and Beyond

IEEE STC
Parallel Model & system
DataFlow and Beyond

Home | News | Timeline | Our Team | Publication | Knowledge Base | STC Event

STC Founding Member

These are the people that made possible the creation of this IEEE Special Technical Community.

- **Arvind**
  Massachusetts Institute of Technology, USA
- **David Abramson**
  The University of Queensland, Australia
- **Georgi Gaydadjiev**
  Maxeler, Imperial College London, UK
- **Guang R. Gao**
  University of Delaware, USA
- **Hirarki Kei**
  The University of Tokyo, Japan
- **Hironori Kasahara**
  Waseda University, Japan
- **Jack B. Dennis**
  Massachusetts Institute of Technology, USA
- **Jean-Luc Gaudiot**
  University of California, Irvine, USA
- **Jin Hai**
  Huazhong University of Science and Technology, China
- **Kuan-Ching Li**
  Providence University, Taiwan
- **Mateo Valero**
  Universitat Politécnica de Catalunya, Spain
- **Mei Hong**
  Beijing Institute of technology, China
- **Nahid Emad**
  University of Versailles, France
- **Nelson Amaral**
  University of Alberta, Canada
- **R. Govindarajan**
  Indian Institute of Science, India
- **Roberto Giorgi**
  Università di Siena, Italy
- **Skevos Evripidou**
  University of Cyprus, Cyprus
- **Stephane Zuckerman**
  Michigan Technological University, USA
- **Vivek Sarkar**
  Rice University, USA
- **Won Woo Ro**
  Yonsei University, South Korea
- **Zheng Weimin**
  Tsinghua University, China

IEEE STC
Parallel Model & system
DataFlow and Beyond

Home | News | Timeline | Our Team | Publication | Knowledge Base | STC Event

Click here to see the STC founding members

Members of our STC

- Dr. Mario Donato Marino
- Dr Lorenzo Verdoscia
- Marco Procaccini
- Dr. Chen Liu
- Jin Yan
- Tongsheng Geng
- Dr. Sreepathi Pai
- Peng Qu
- Dr. Albert Cohen
- Shipeng Qi
- Dr. Toshiaki Kitamura
- Jose M Monsalve Diaz
- Dr Erik Altman
- Dr. Karu Sankaralingam
- Dr. Jidong Zhai
- Professor Eduardo Juarez

- Dr. Andres Marquez
- Yoshitake Oki
- Professor Junqing Yu
- Professor wenguang chen
- Professor Sunita Chandrasekaran
- Professor Dongrui Fan
- Joshua Suetterlein
- Professor Bob Iannucci
- Dr Long Zheng
- Dr. Hao Tu
- Dr Tobias Becker
- Siddhisanket Raskar
- Professor Marco Aldinucci
- Professor Avi Mendelson
- Professor Xiaoming Li
- Dr. Xu Tan

- Data Stream Processing (DaSP)

- Real-time processing of continuous data streams
- Processed on-the-fly with stringent Quality of Service (QoS)
- Potentially irregular flows of data must be timely processed
  - detect anomalies, real-time incremental responses, etc.

Stream processing applications

D. Turaga, H. Andrade, B. Gedik, et al. Design principles for developing stream processing applications. Softw. Pract. Exper. 2010; **40**:1073–1104

INTERSECTING DIFFERENT CS AREAS

- Control Systems (and CPS)
  - Including network processors, FPGAs, etc
- Parallel Computing (shared memory)
  - StreamIt (2006), TBB, FastFlow (2009), RaftLib (2013), StreamBox (2016), …
- BigData Analytics
  - Lambda, Kappa architectures
  - Apache + **x** ∈ {Kafka, Flink, Storm, Apex, Spark, … and counting}
  - Tensorflow (2015)

APACHE Spark Streaming

Download | Libraries | Documentation | Examples | Community | Developers | Apache Software Foundation

- SQL and DataFrames
- Spark Streaming
- MLlib (machine learning)
- GraphX (graph)
- Third-Party Projects

S kes it easy to build scalable fault-tolerant streaming applications.

Ease of Use

Build applications through high-level operators.

Spark Streaming brings Apache Spark's language-integrated API to stream processing, letting you write streaming jobs the same way you write batch jobs. It supports Java, Scala and Python.

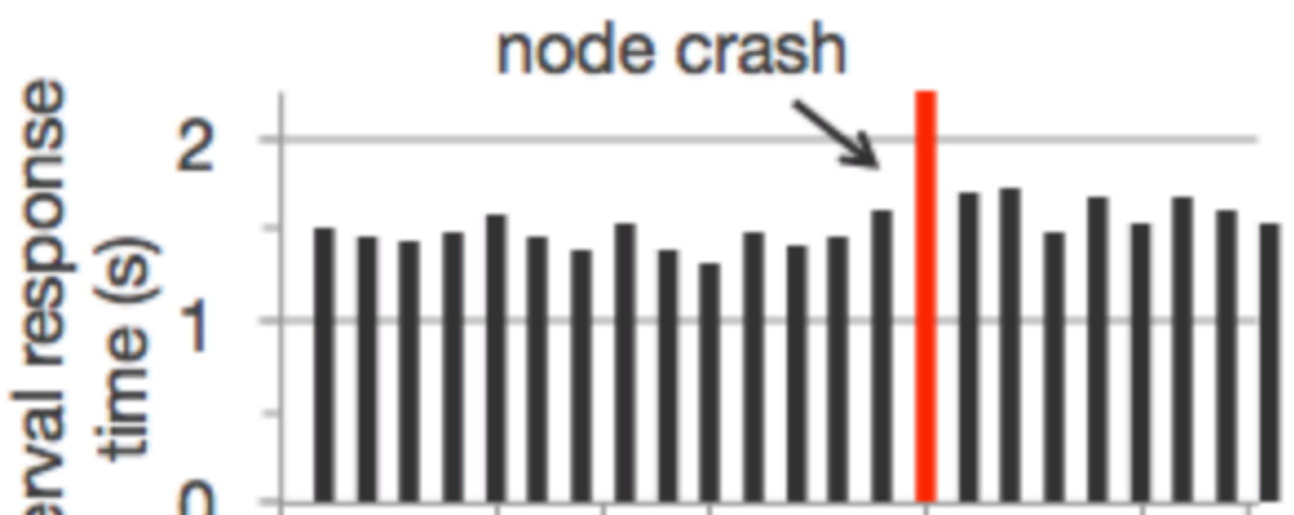
```
TwitterUtils.createStream(...)
    .filter(_.getText.contains("Spark"))
    .countByWindow(Seconds(5))
```

Counting tweets on a sliding window

Fault Tolerance

Stateful exactly-once semantics out of the box.

Spark Streaming recovers both lost work and operator state (e.g. sliding windows) out of the box, without any extra code on your part



Latest News

Spark 2.2.0 released (Jul 11, 2017)

Spark 2.1.1 released (May 02, 2017)

Spark Summit (June 5-7th, 2017, San Francisco) agenda posted (Mar 31, 2017)

Spark Summit East (Feb 7-9th, 2017, Boston) agenda posted (Jan 04, 2017)

Archive

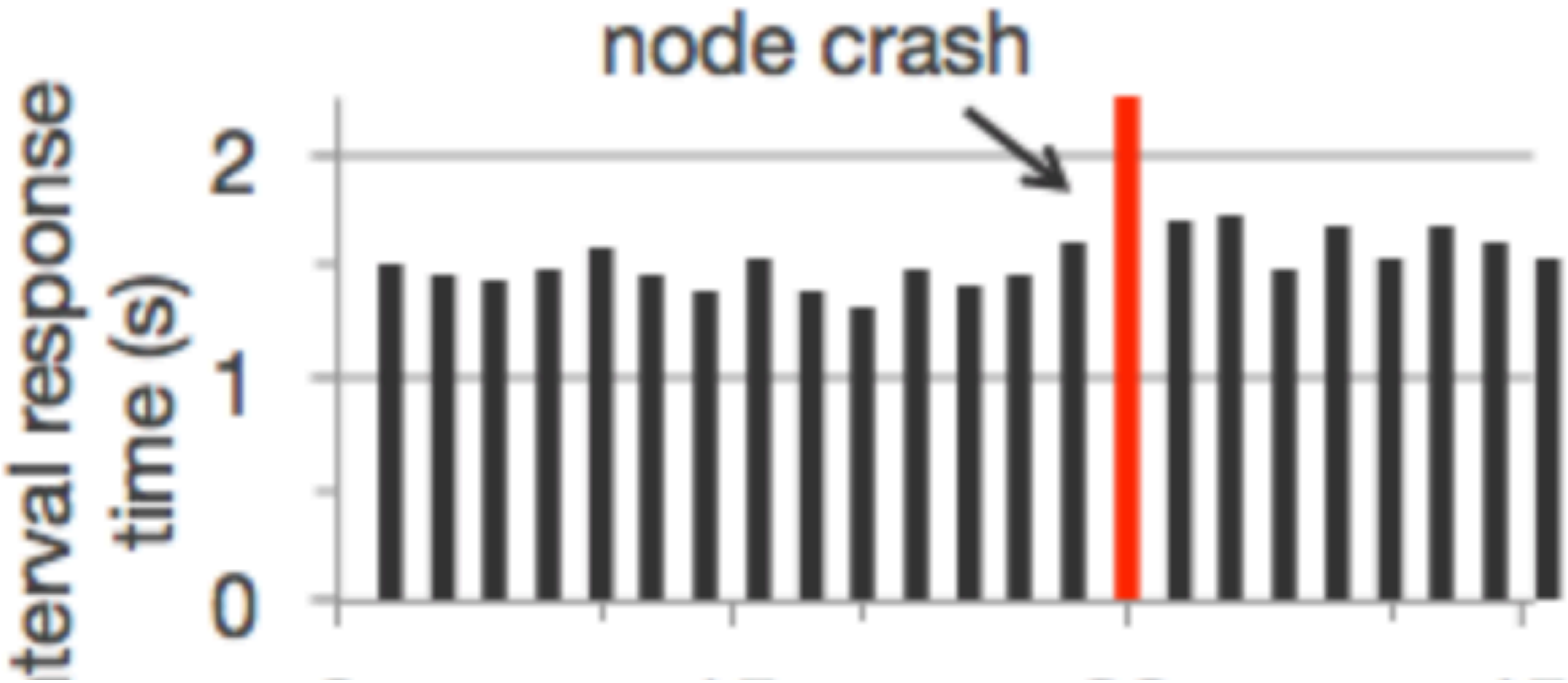
Download Spark

Built-in Libraries:

- SQL and DataFrames
- Spark Streaming
- MLlib (machine learning)
- GraphX (graph)
- Third-Party Projects

eam
ch

Counting tweets on a sliding window



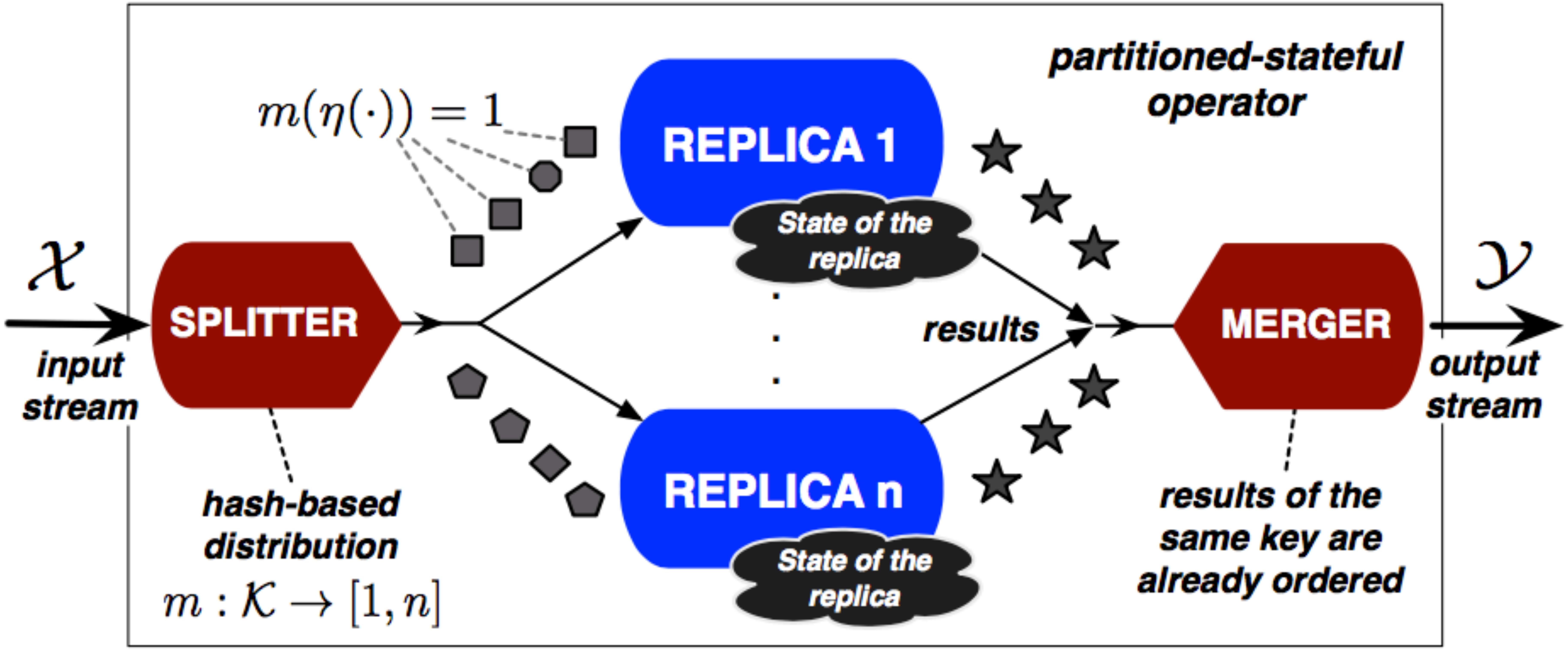
Dow

Built-in Libra

SQL and DataF
Spark Streamin
MLlib (machine
GraphX (graph)

Third-Party Pro

STREAMING: CORE PARADIGM



Tiziano De Matteis and Gabriele Mencagli. Keep Calm and React with Foresight: Strategies for Low- Latency and Energy-Efficient Elastic Data Stream Processing, PPoPP 2016.

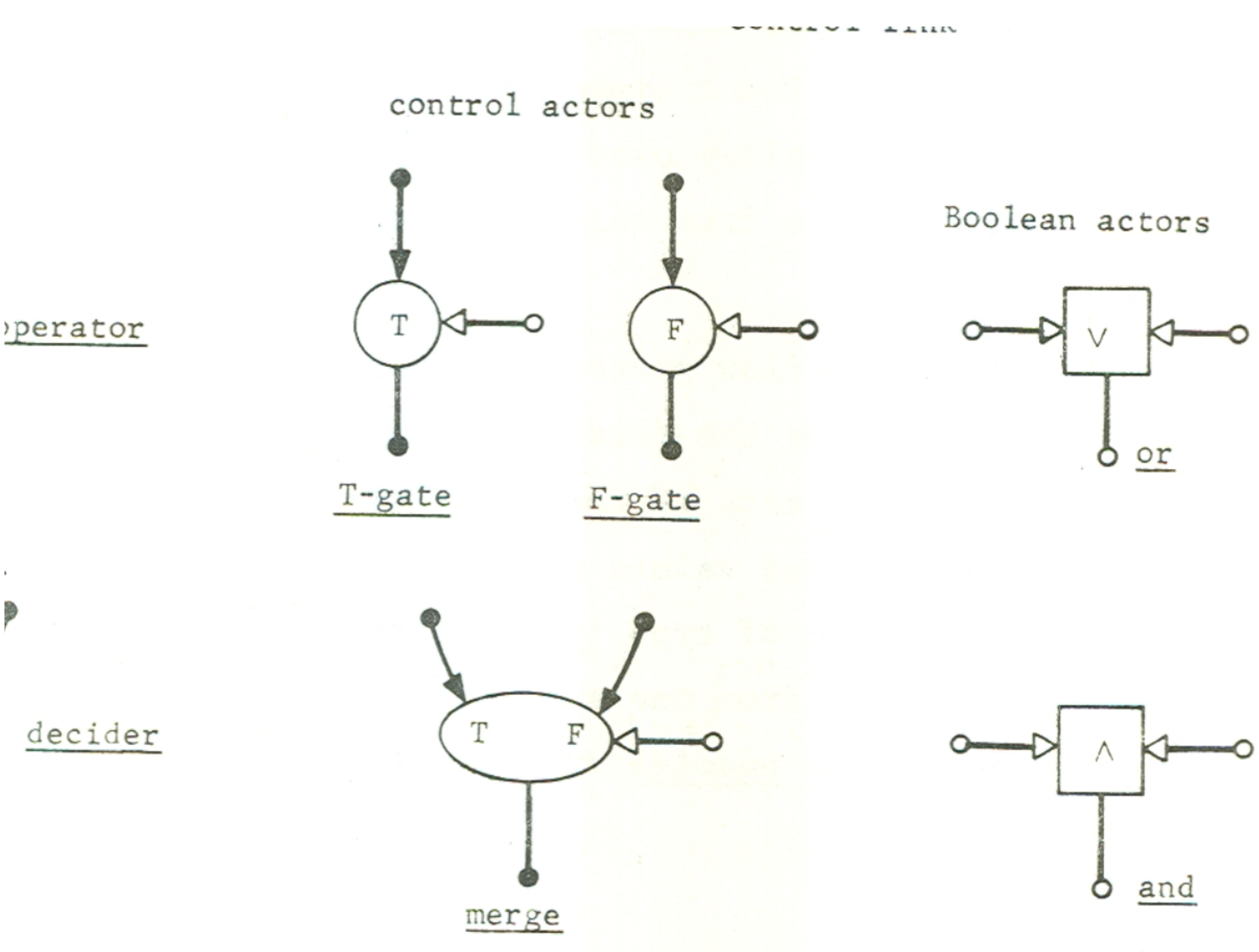
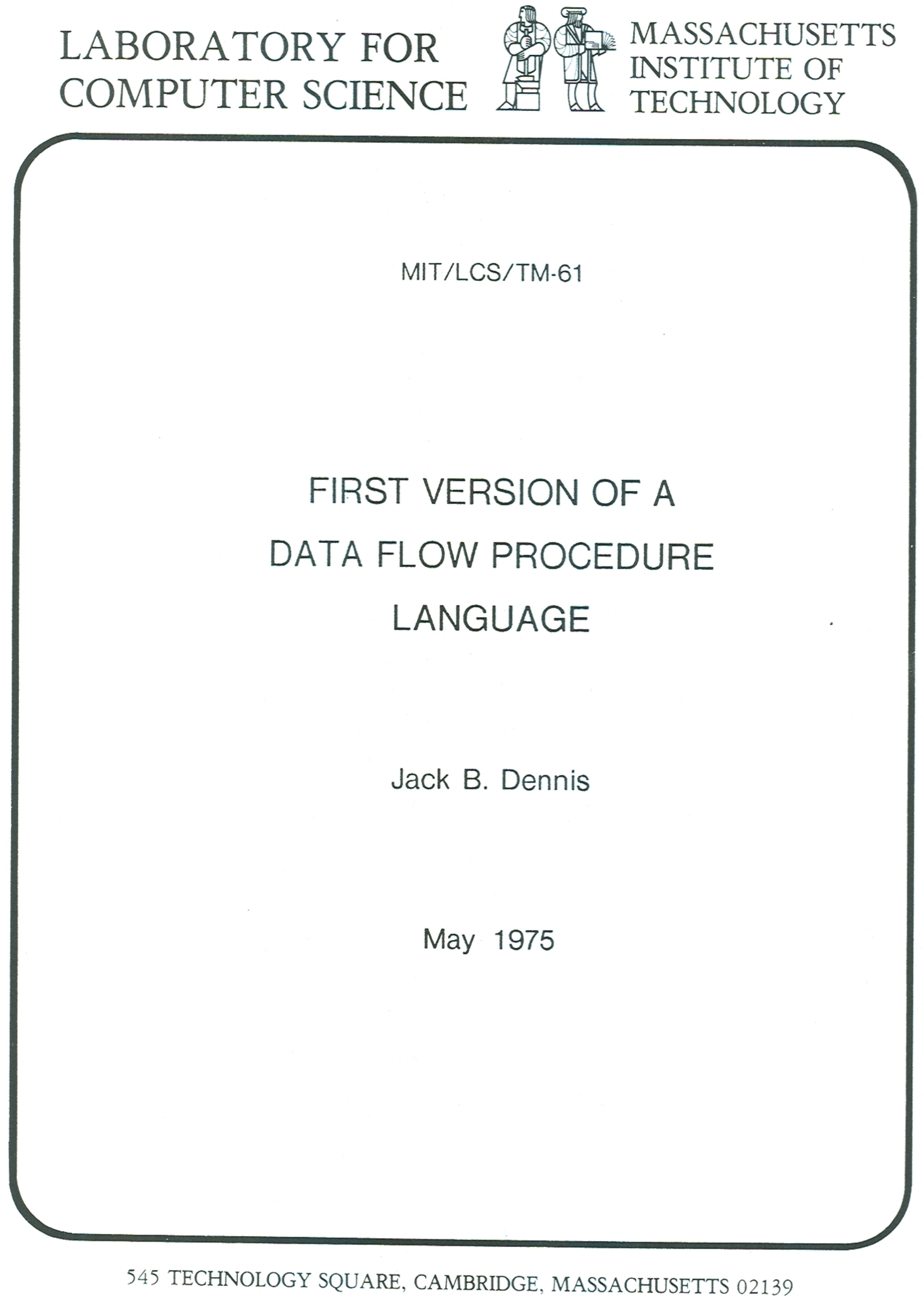


Figure 2. Node types for data flow programs.


LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TM-61

FIRST VERSION OF A DATA FLOW PROCEDURE LANGUAGE

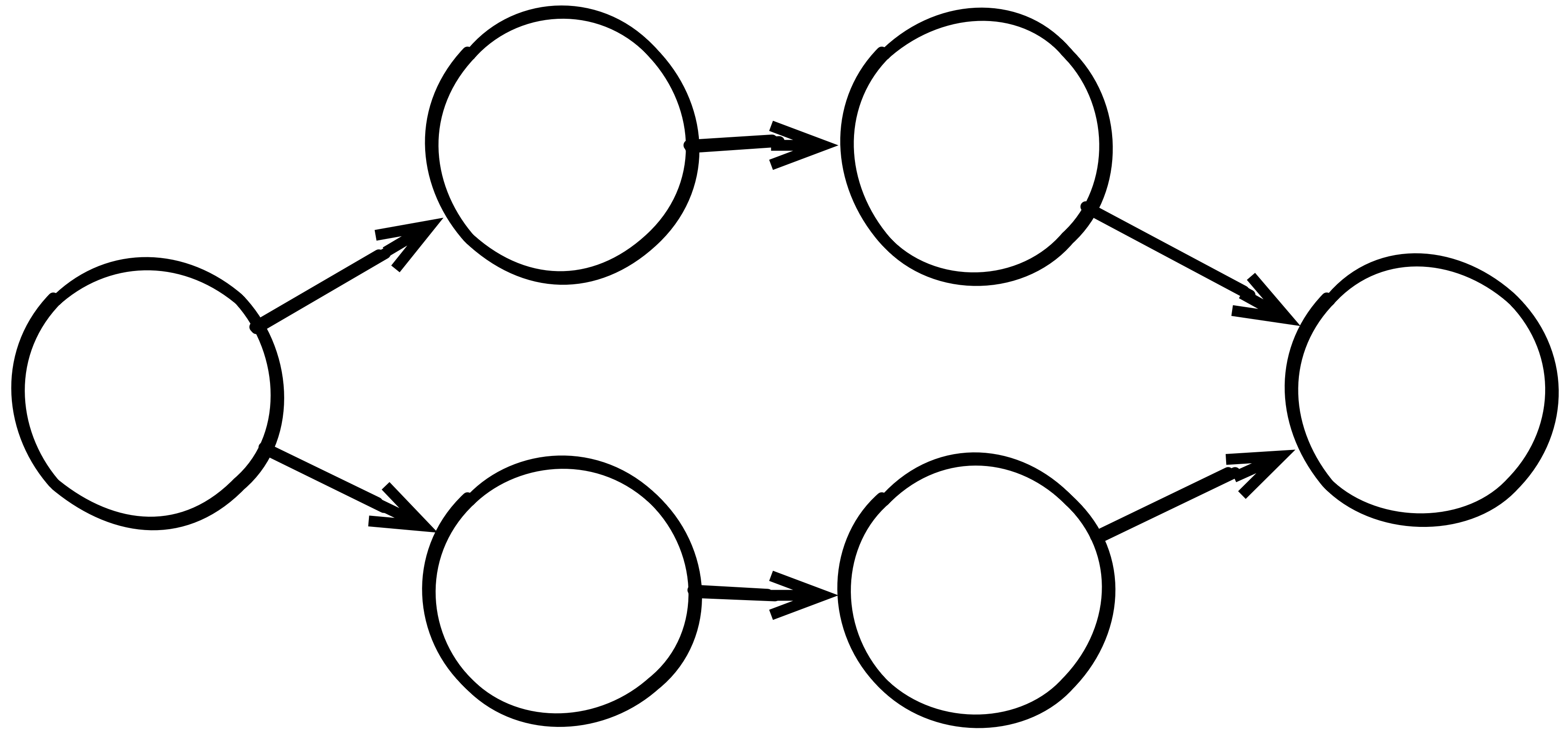
Jack B. Dennis

May 1975

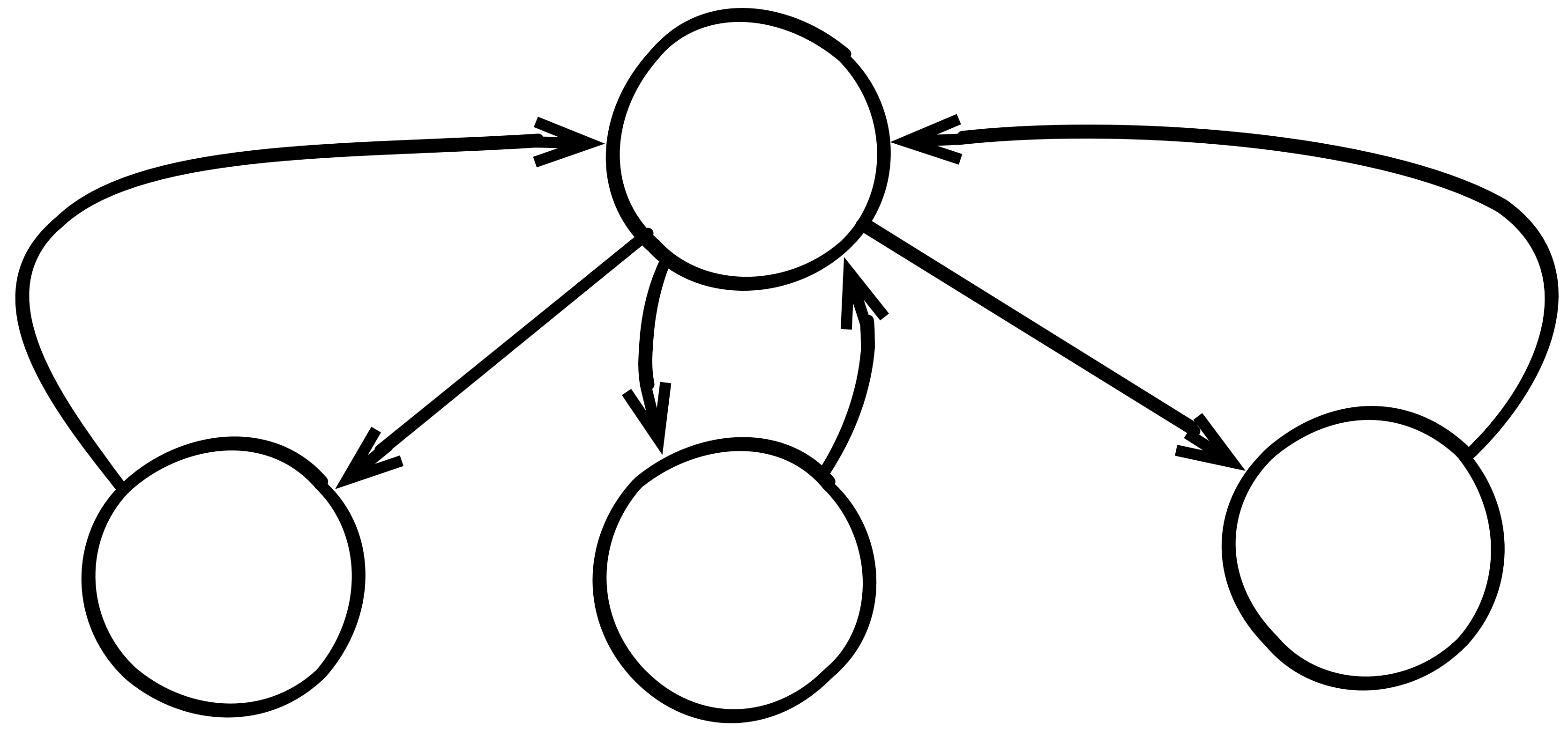
545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Network of executors vs Task graph

- Graph of tasks
  - Dataflow — typically DAG
  - Each node is a task
  - E.g. a C++ object
  - Problems: firing, scheduling, etc.



- Network of executors
  - “Controlflow" — typically cyclic graph
  - E.g. threads or processes
  - Problems: pinning, mapping, pooling, etc.

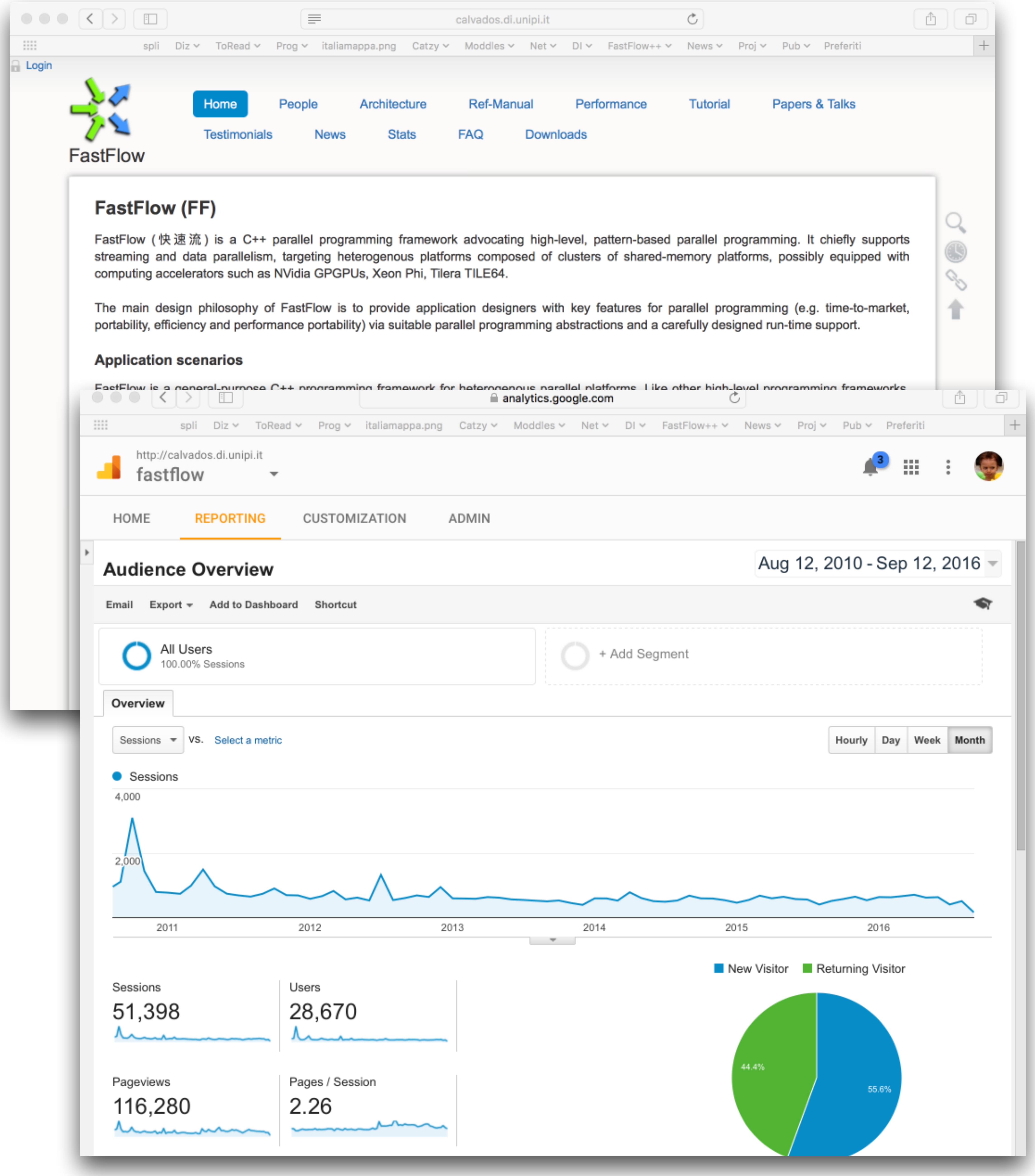


FastFlow

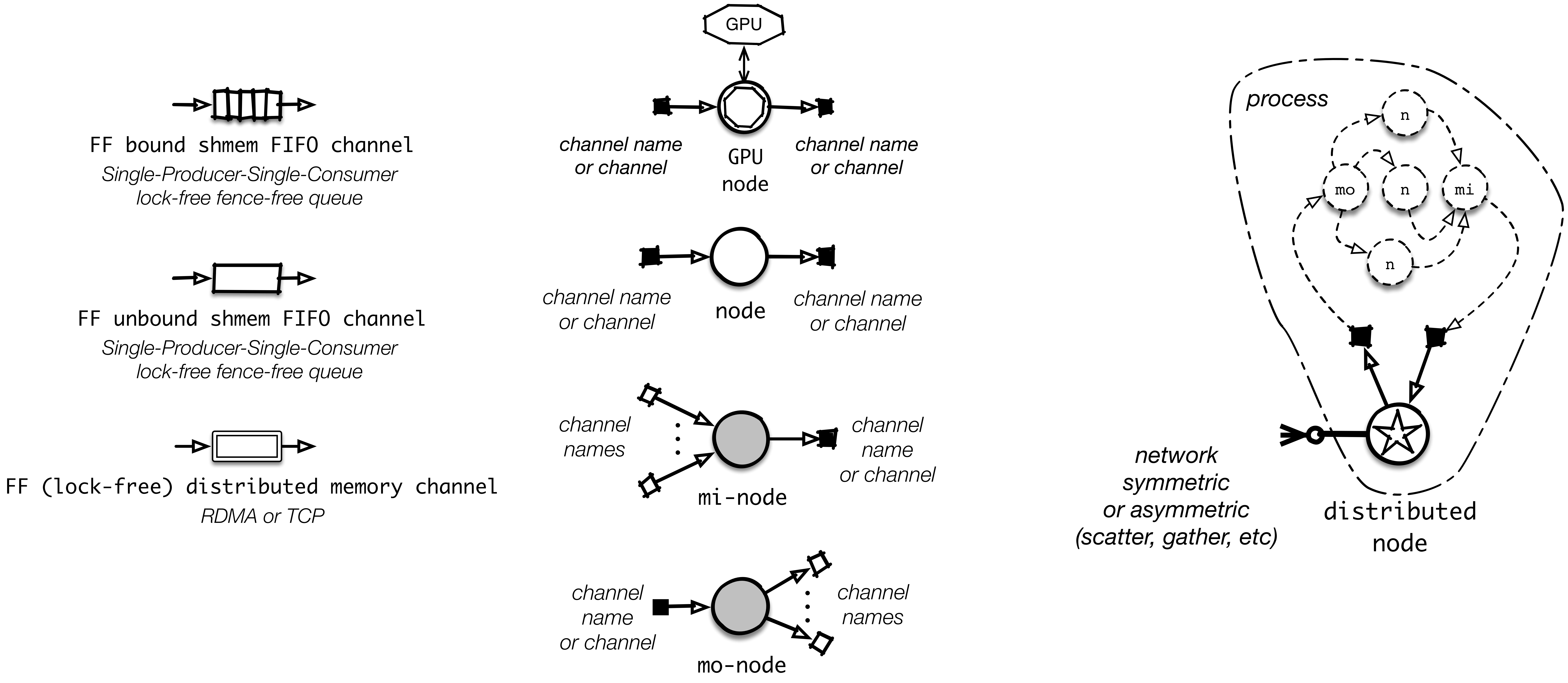
http://mc-fastflow.sourceforge.net/

- **Toreador** (EC-RIA, H2020, ICT-16-2015 big data): TrustwOrthy model-awaRE Analytics Data platfORm (2016, 36 months, total cost 6.5M €)
- **Rephrase** (EC-RIA, H2020, ICT-2014-1): Refactoring Parallel Heterogeneous Resource-Aware Applications – a Software Engineering Approach (2015, 36 months, total cost 3.5M €)
- **HyVar** (EC-RIA, H2020, ICT-2014-1): Scalable Hybrid Variability for Distributed Evolving Software Systems (2015, 36 months, total cost 2.8M €)

- **REPARA** (EC-STREP, 7th FP): Reengineering and Enabling Performance And poweR of Applications (2013, 36 months, total cost 3.5M €)
- **ParaPhrase** (EC-STREP, 7th FP): Parallel Patterns for Adaptive Heterogeneous Multicore Systems (2011, 42 months, total cost 4.2M €)
- **IBM Research** 3 faculty awards 2015 (50K $)
- **Noesis Solutions**: Machine learning for engineering 2015 (75K €)
- **A3CUBE Inc.**: FastFlow/PGAS with in memory fabric 2014
- **NVidia Corp**: CUDA Research Center at University of Torino 2013



SWSR QUEUES + MEDIATORS



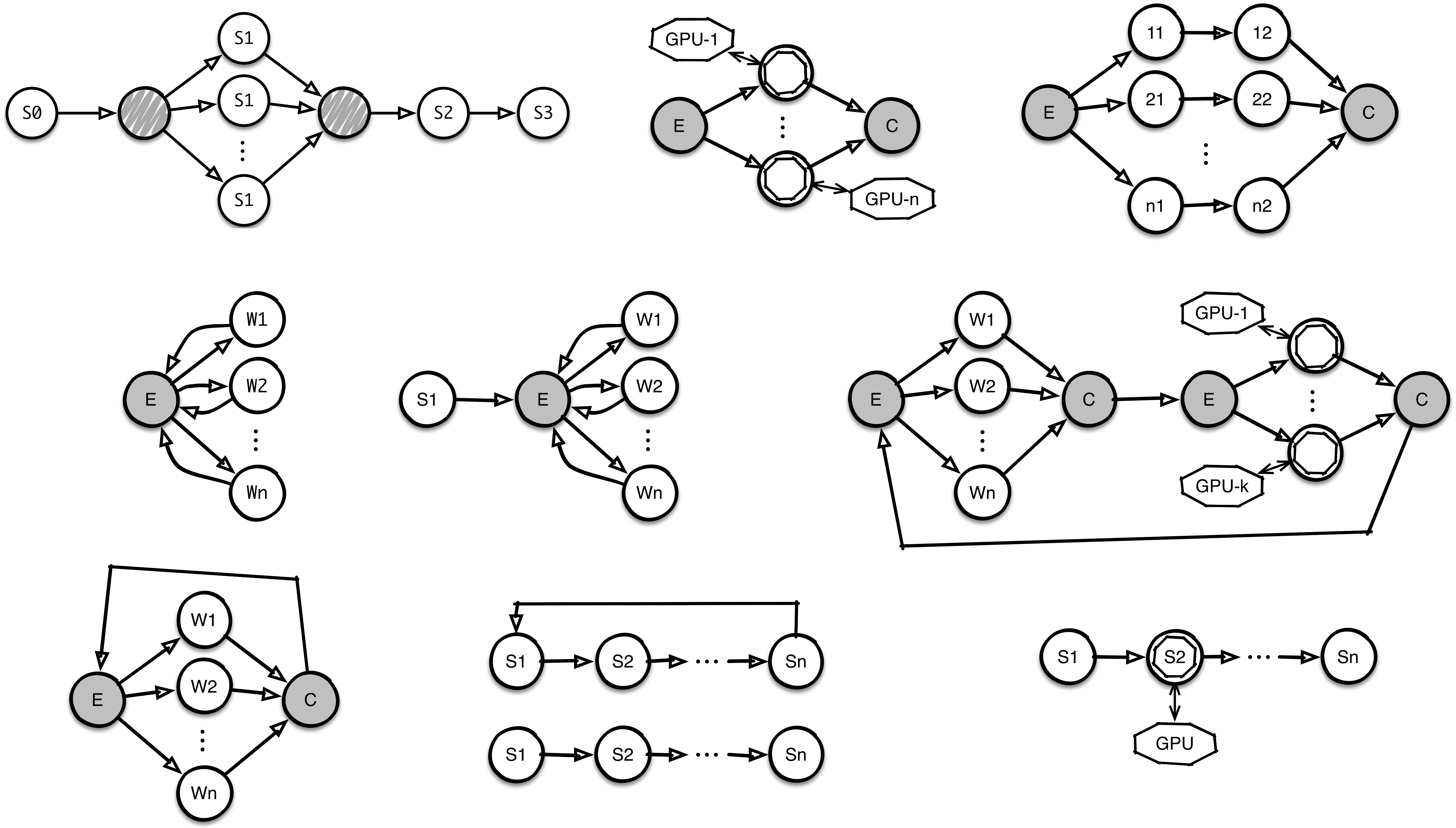
Generate the network

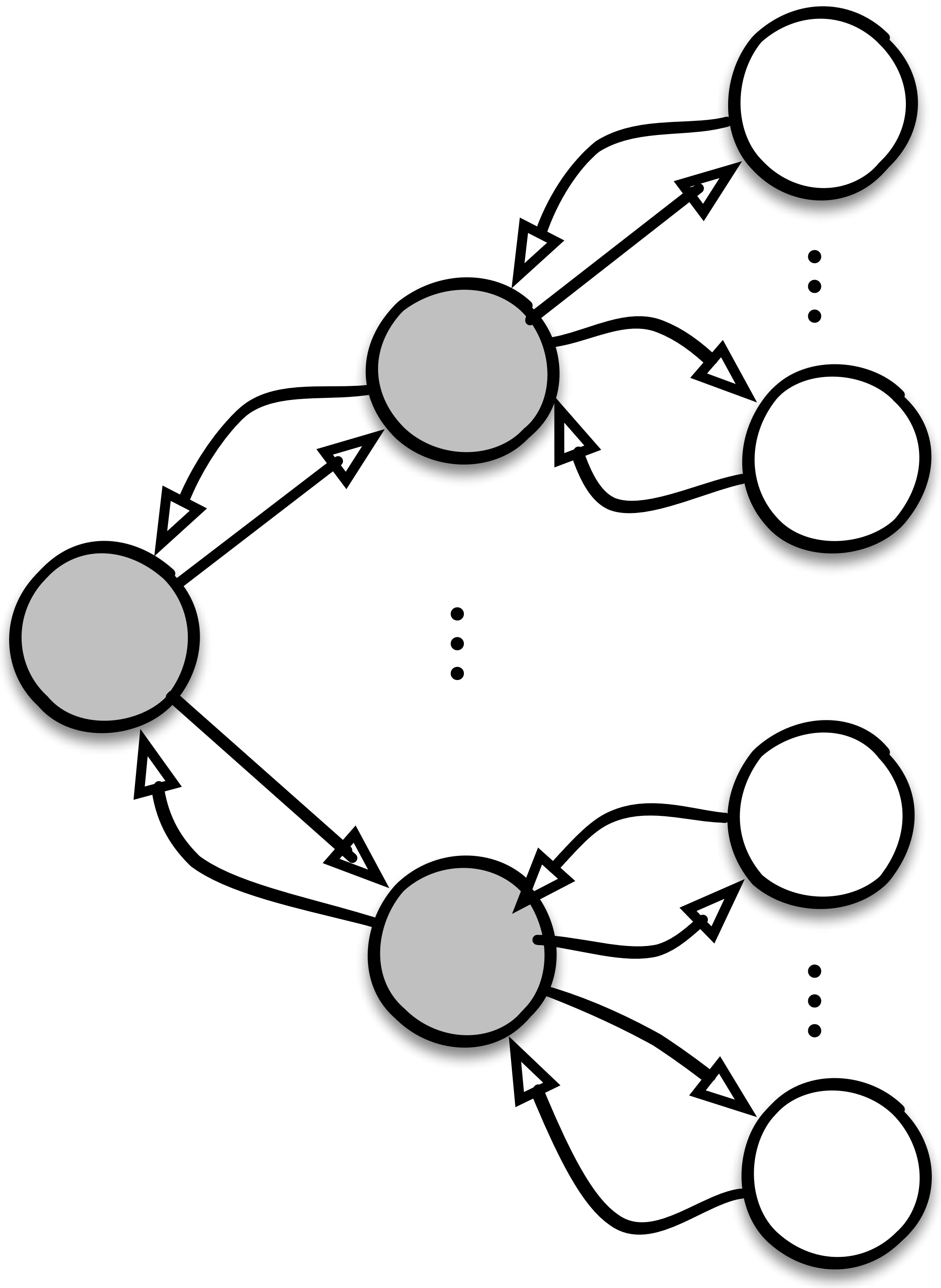
True data dependencies moves across arrows

Composing via mediator guarantee
correctness (data races & deadlock freedom)

Generate the network

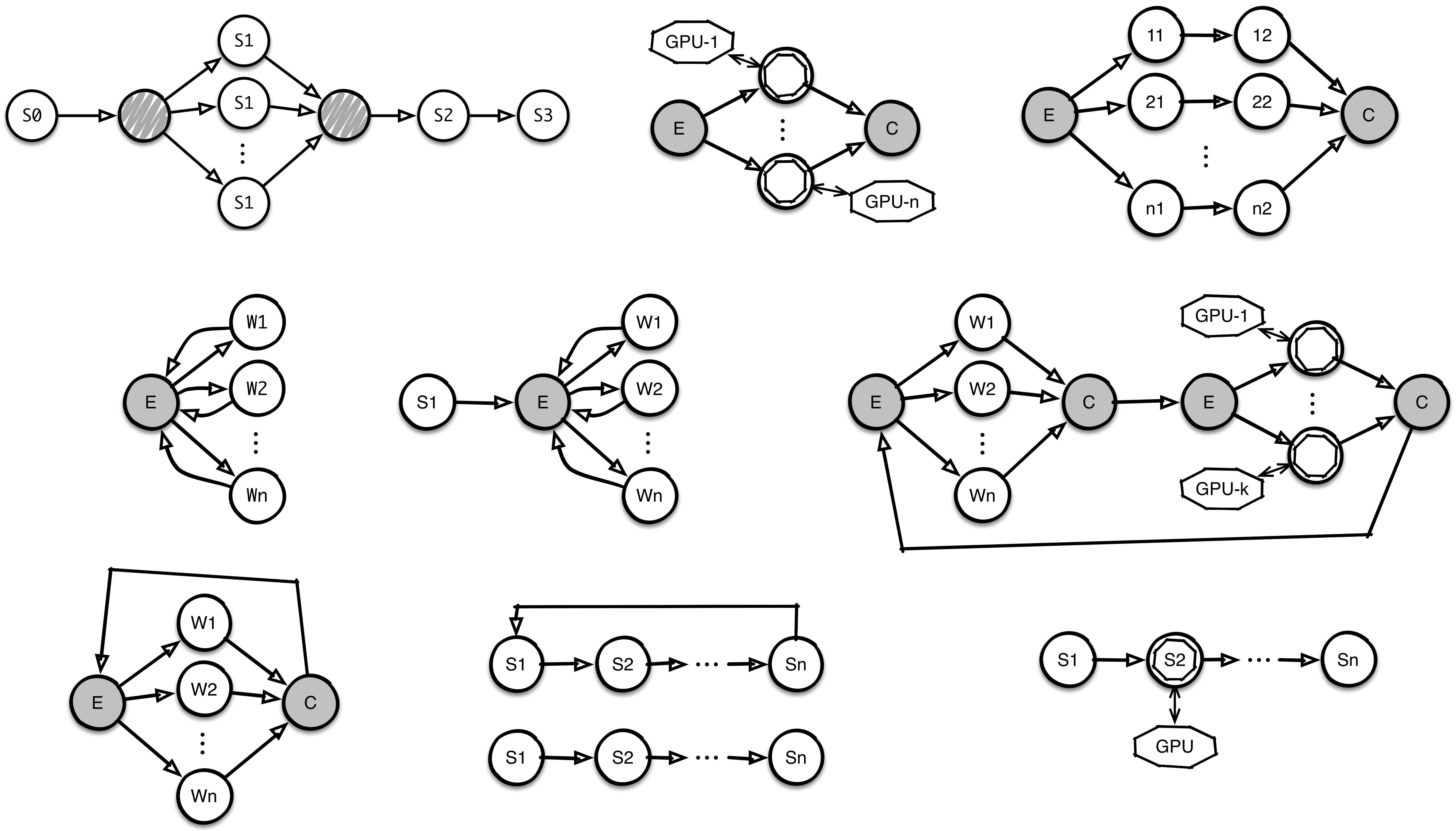
True data dependencies moves across arrows

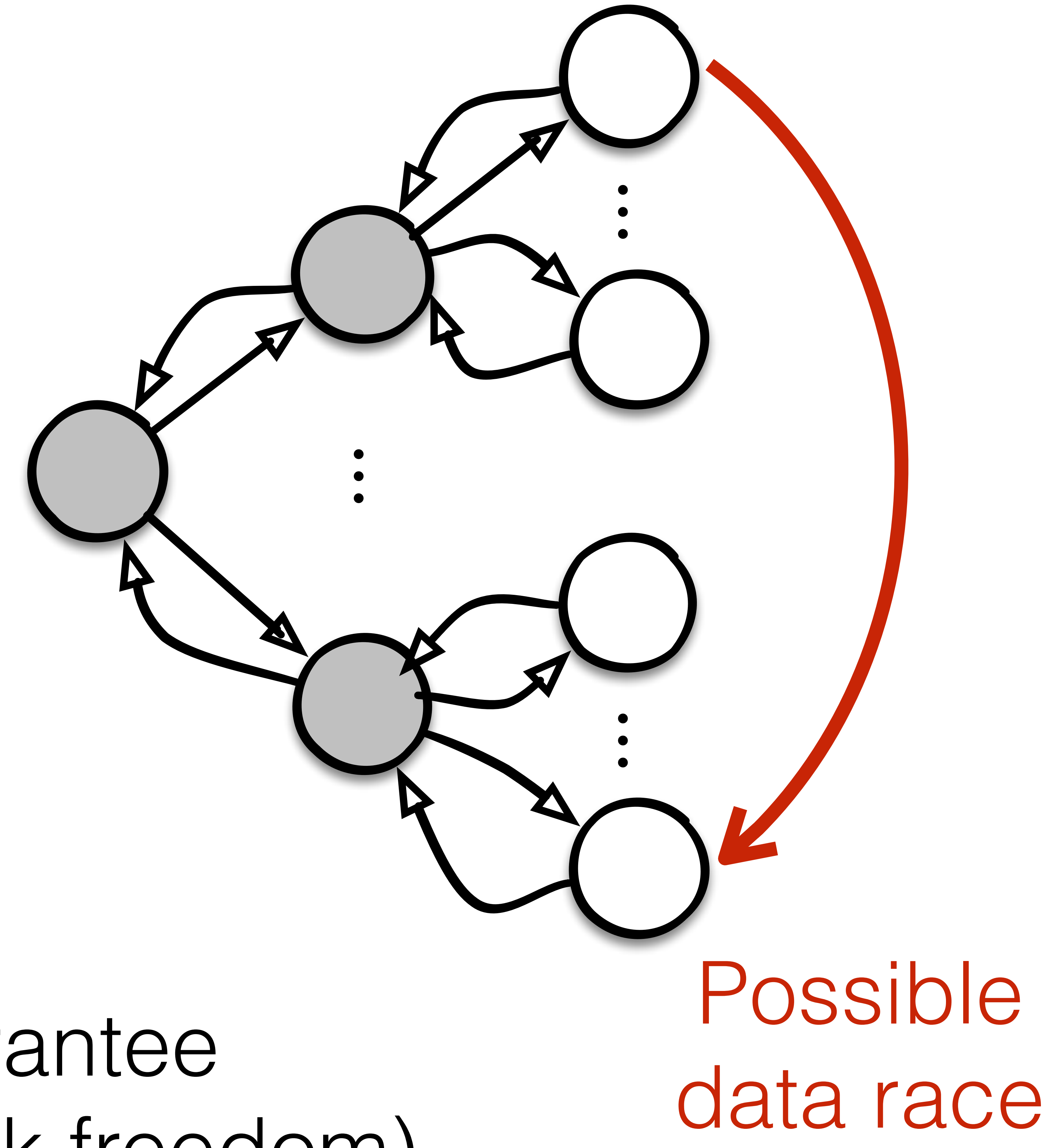




Composing via mediator guarantee
correctness (data races & deadlock freedom)

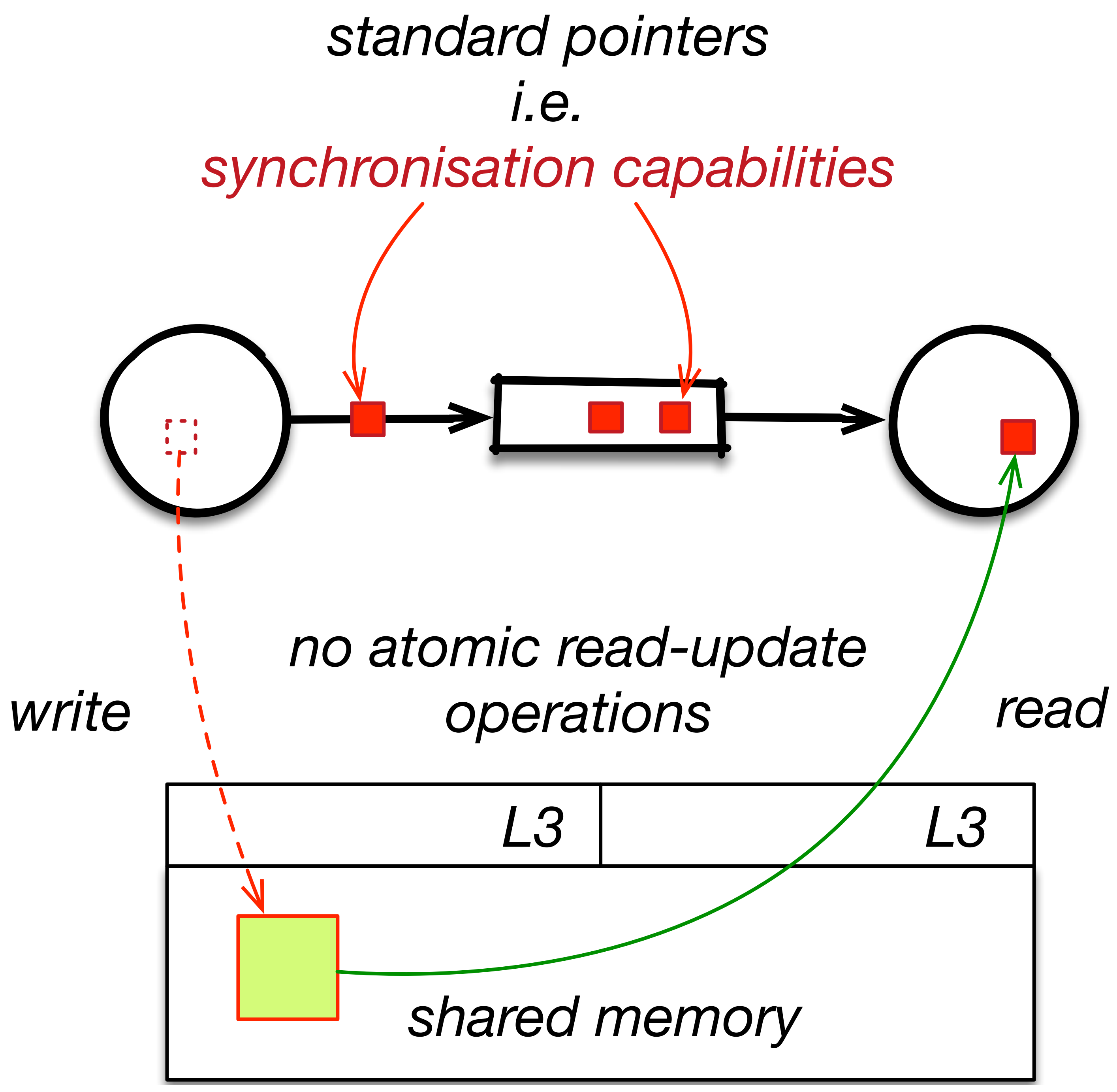
Generate the network
True data dependencies moves across arrows





Composing via mediator guarantee
correctness (data races & deadlock freedom)

Programming model: synchronisations happen by way of P2P data dependencies (thus no atomics are needed)

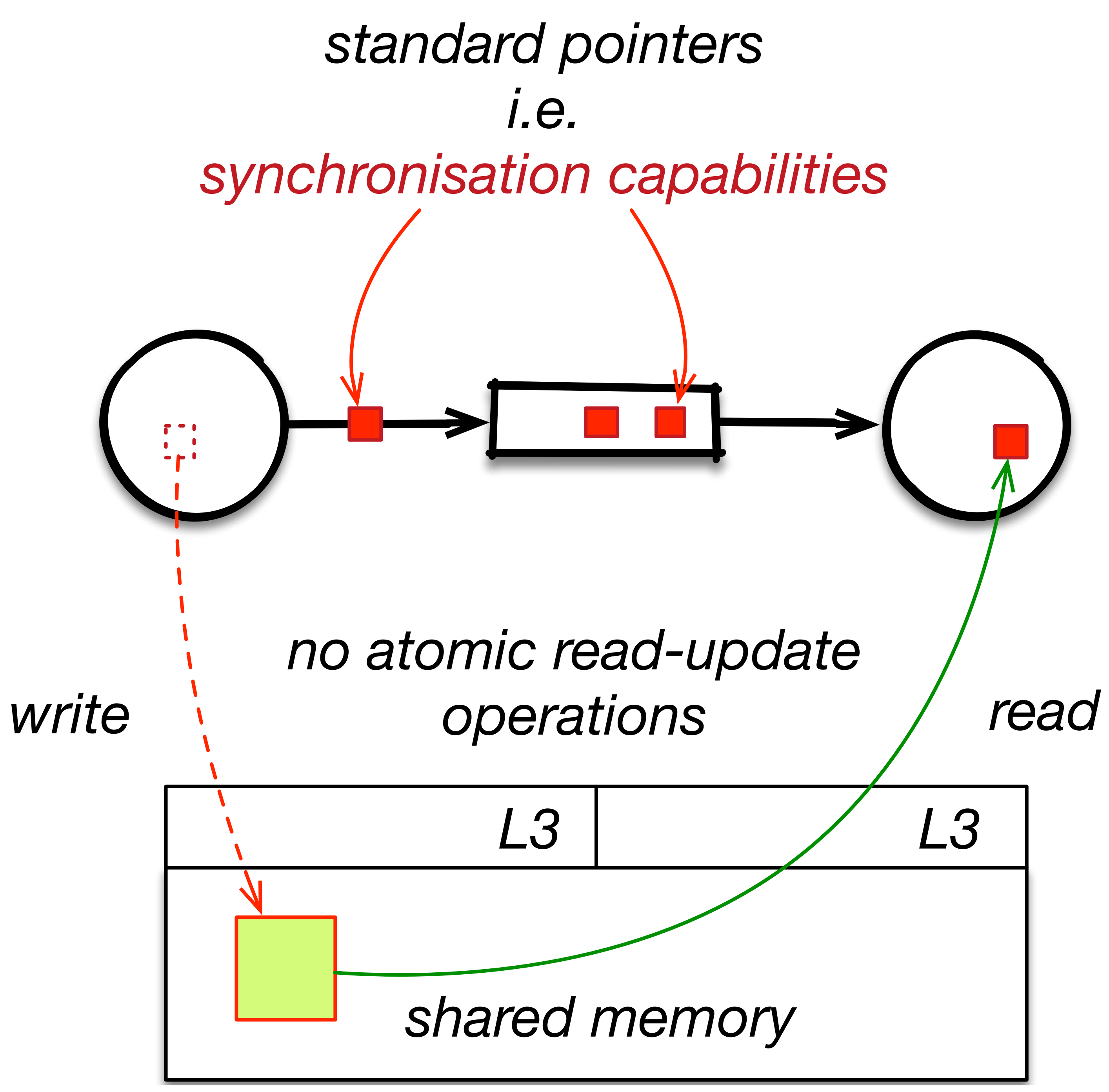


Shared-memory cache-coherent or
non-coherent multicore

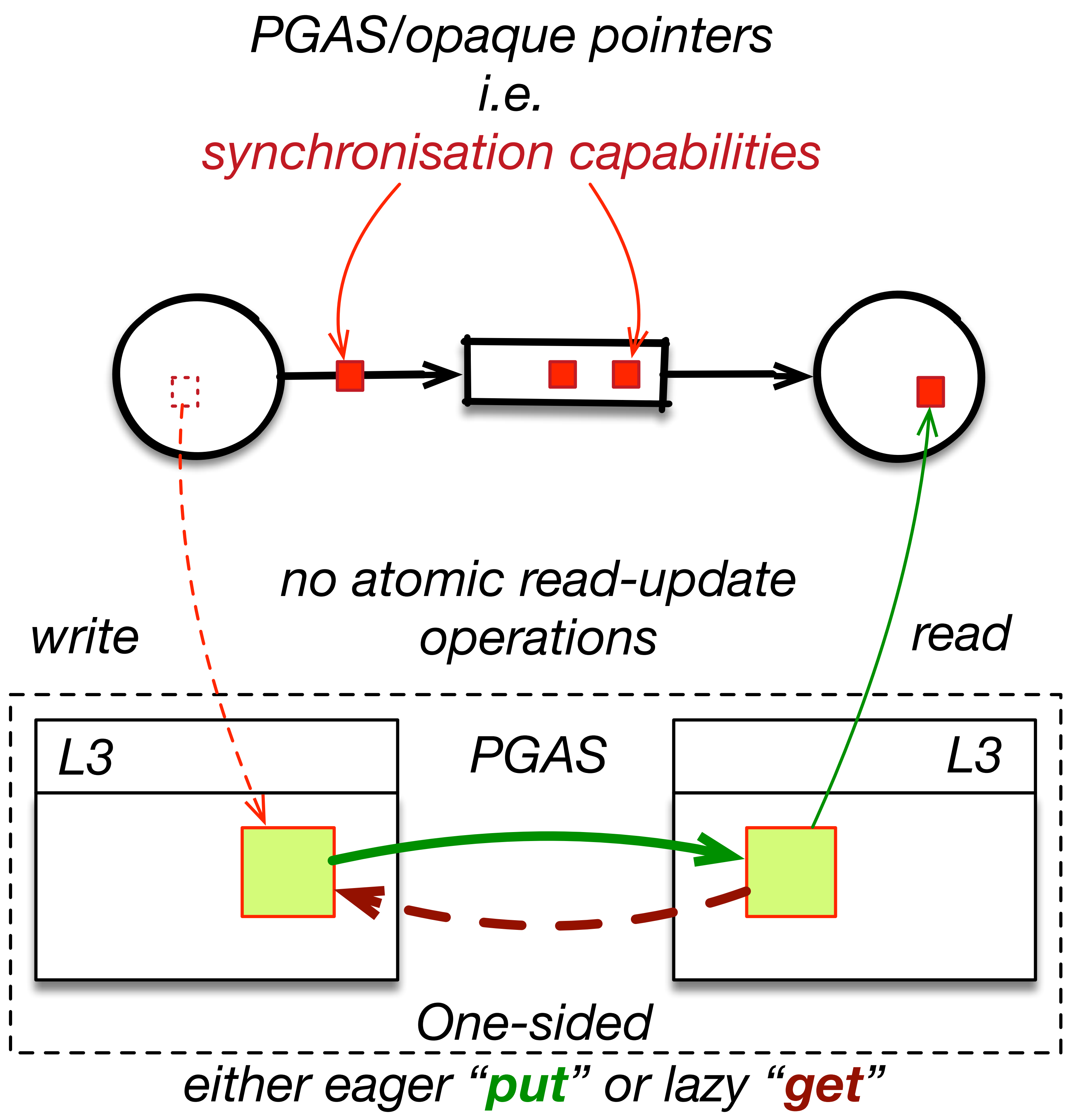
Synchronisations are in a message-passing style, but designers are not forced to think in a distributed way

No copies are needed, the memory fences are but asynchrony helps

Programming model: synchronisations happen by way of P2P data dependencies (thus no atomics are needed)



Shared-memory cache-coherent or non-coherent multicore

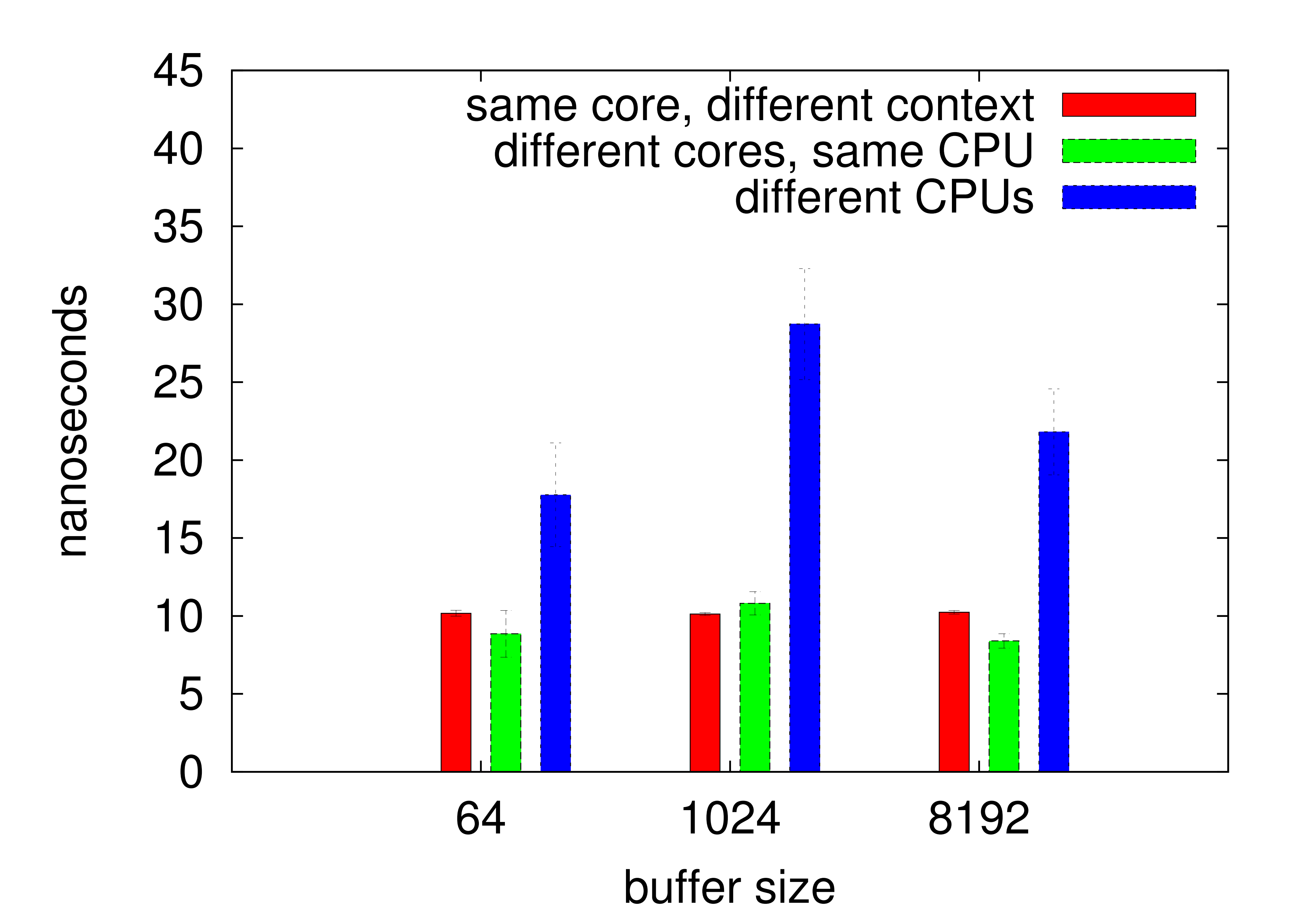


Distributed GAM

BASED ON SINGLE-WRITER-SINGLE-READER FIFO

- Does not require atomic
  - No fence under TSO, WriteFence under WO
    - *J. Giacomoni et al. Fastforward for efficient pipeline parallelism: a cache-optimized concurrent lock-free queue. PPoPP 08*
    - *M. Aldinucci et al. An Efficient Unbounded Lock-Free Queue for Multi-core Systems. Euro-Par 2012*
- Enough to support Producer-Consumer
  - Inherently asynchronous
  - Powerful enough to build a general purpose parallel programming model

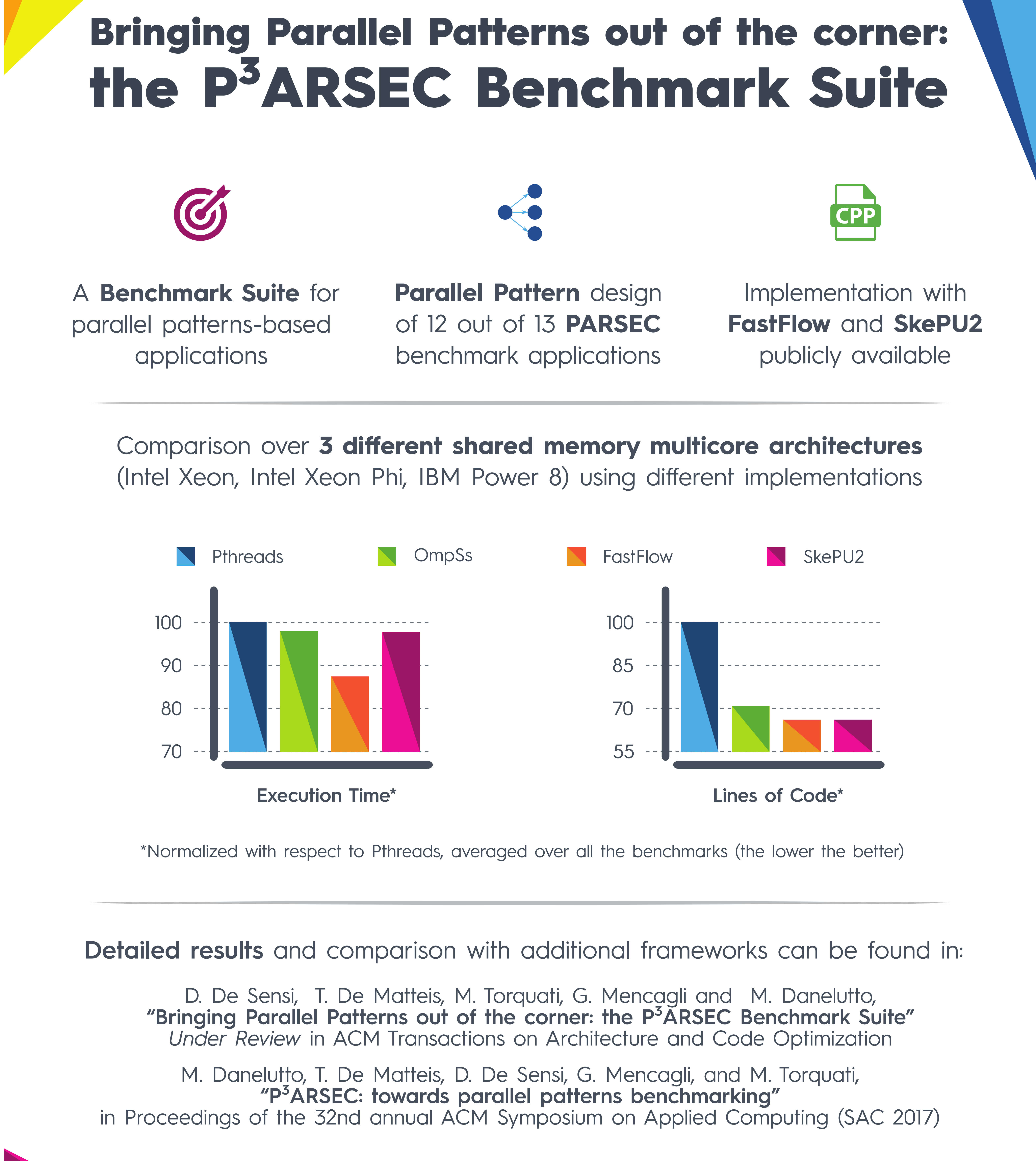
FastFlow unbound queue
core-to-core message latency
Xeon E7-4820 @2.0GHz Sandy Bridge



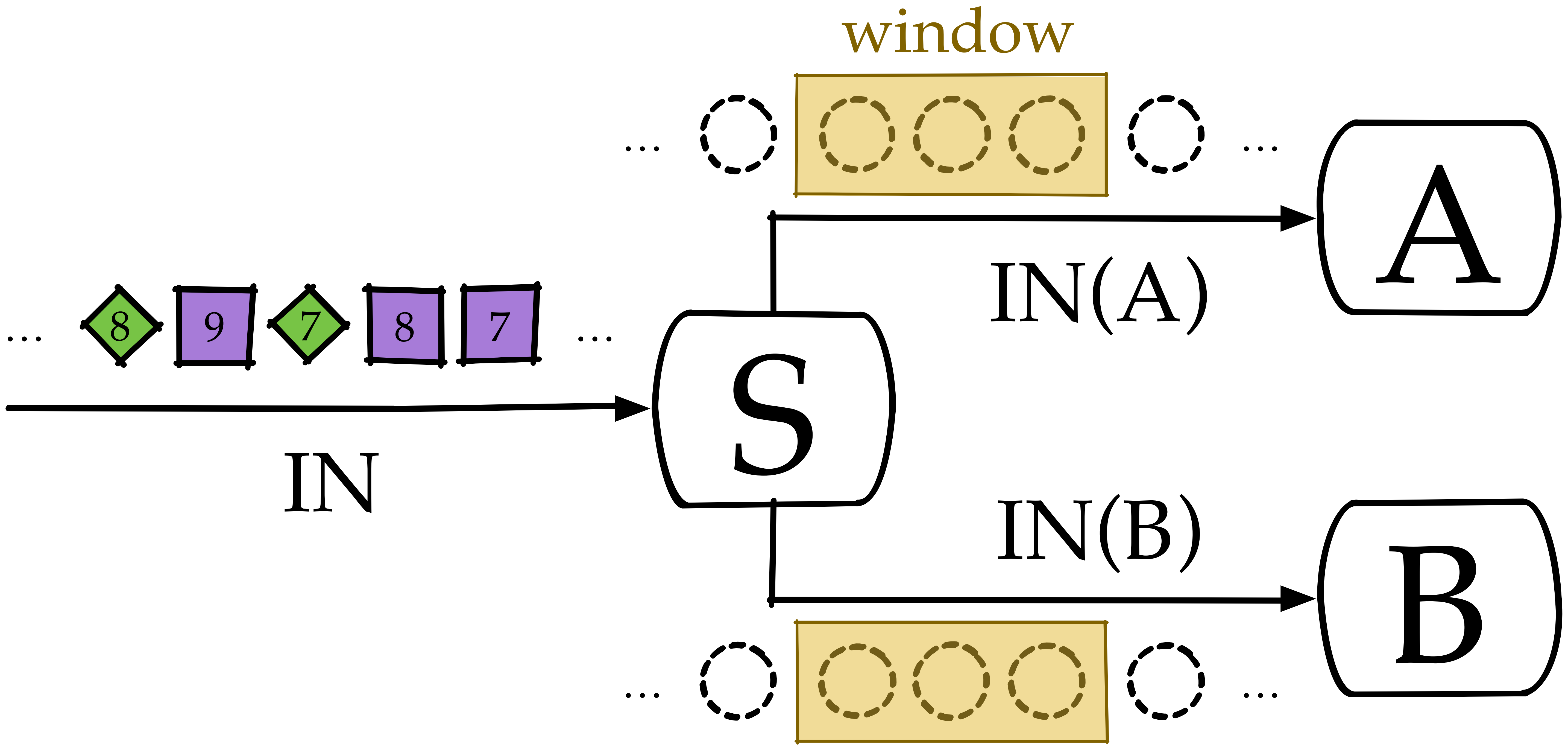
FASTFLOW

http://calvados.di.unipi.it/paragroup/

- Compared in performance and features against: openMP, TBB, Pthreads, OpenSs, MPI, …
- M. Aldinucci, M. Meneghin, and M. Torquati, “Efficient Smith-Waterman on multi-core with FastFlow,” Euromicro PDP 2010.
- 80+ papers from 2010

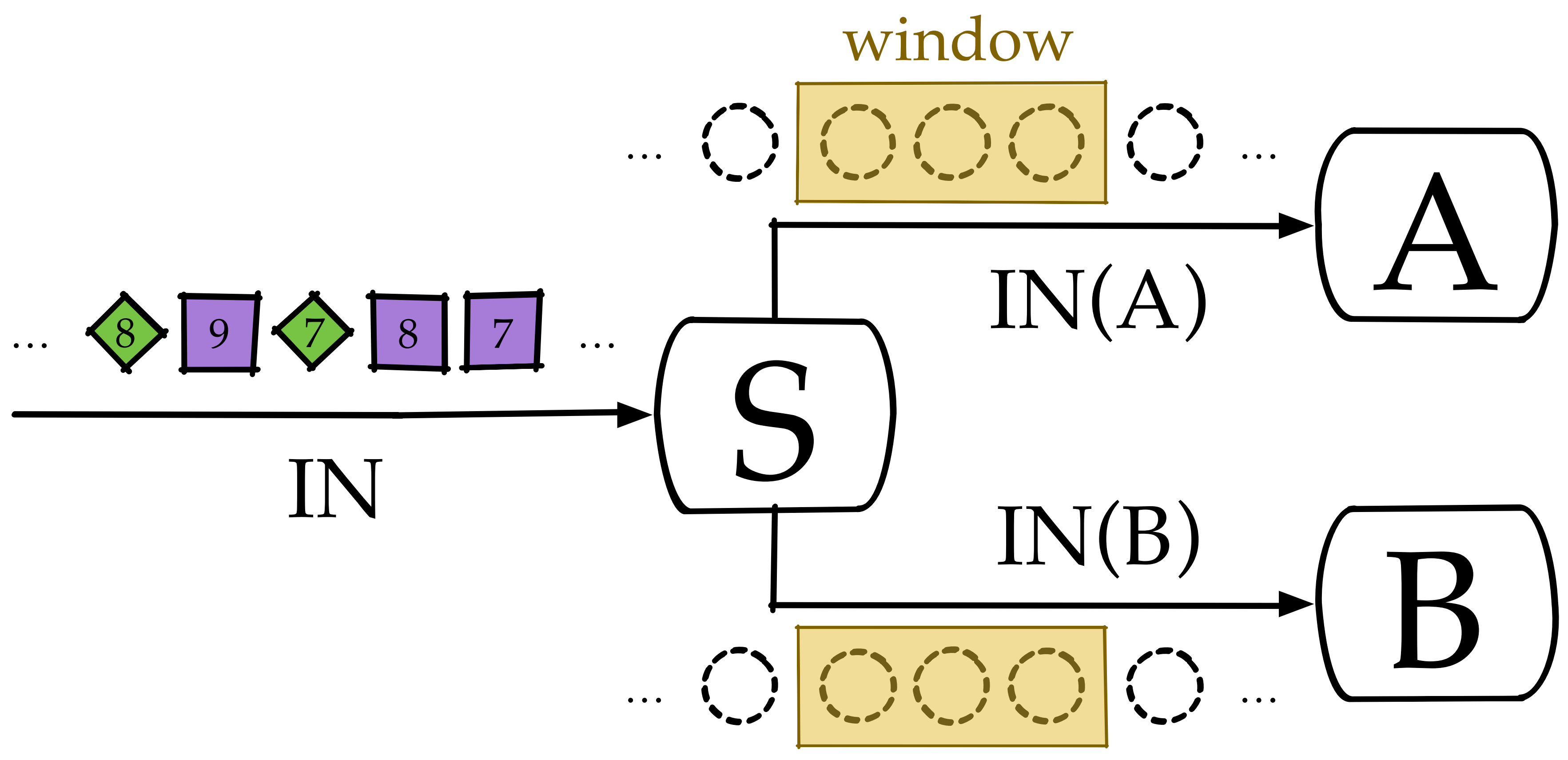


Windowed Stream Processing

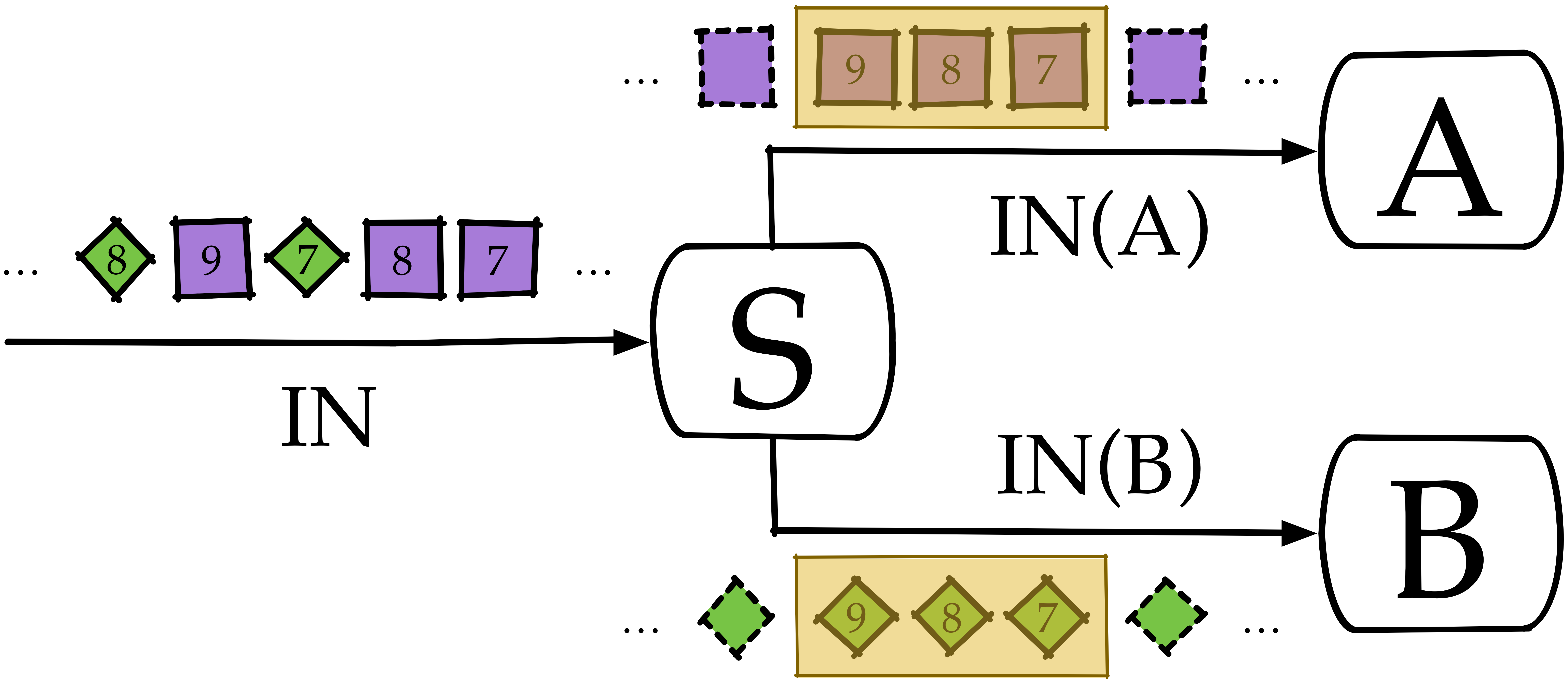


WINDOWED STREAM PROCESSING

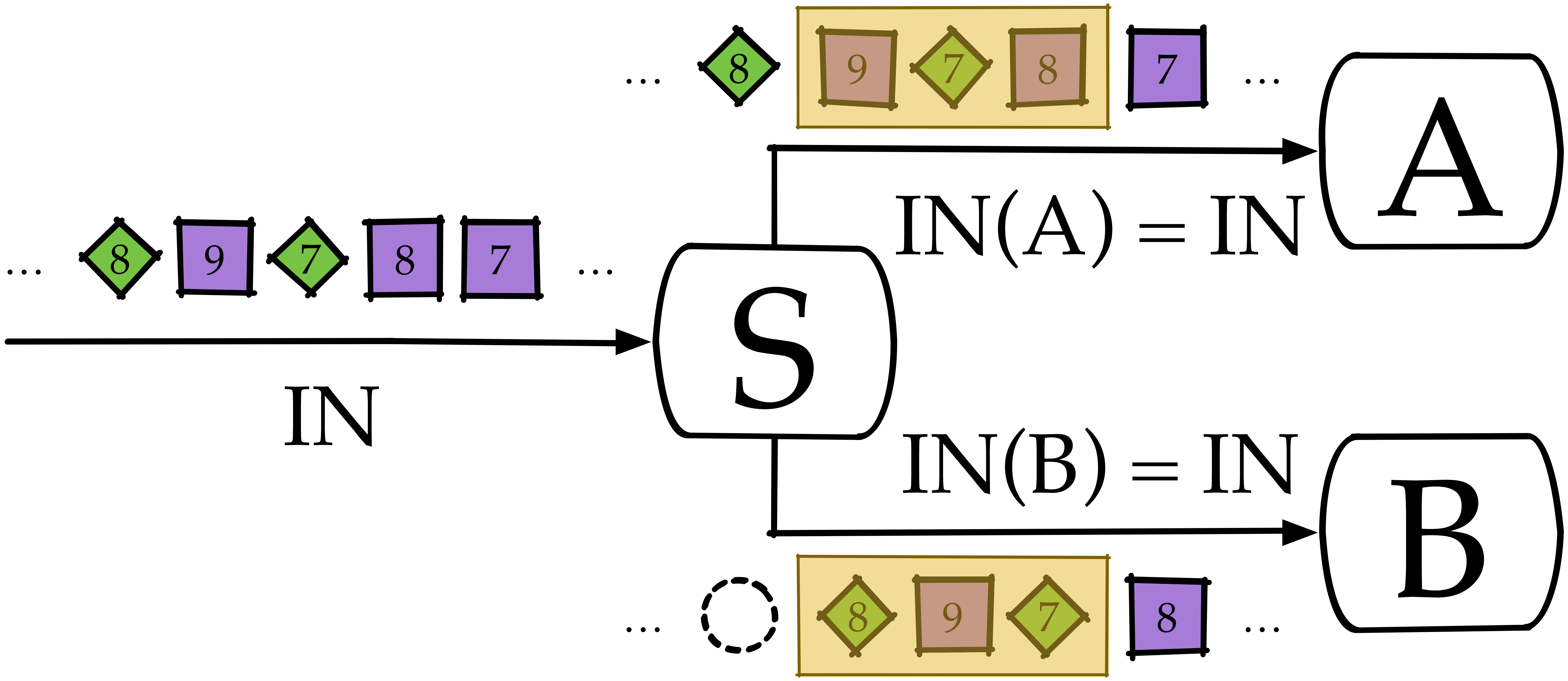
- Windows approximate infinite stream history
  - tuple significance is often time-decaying
  - only the most recent tuples are kept
- Different windowing policies
  - Sliding windows: window size + sliding factor
  - Session windows [Apache Flink, Apache Beam…]
- Common implementation = Worker-side windowing (more parallelism)



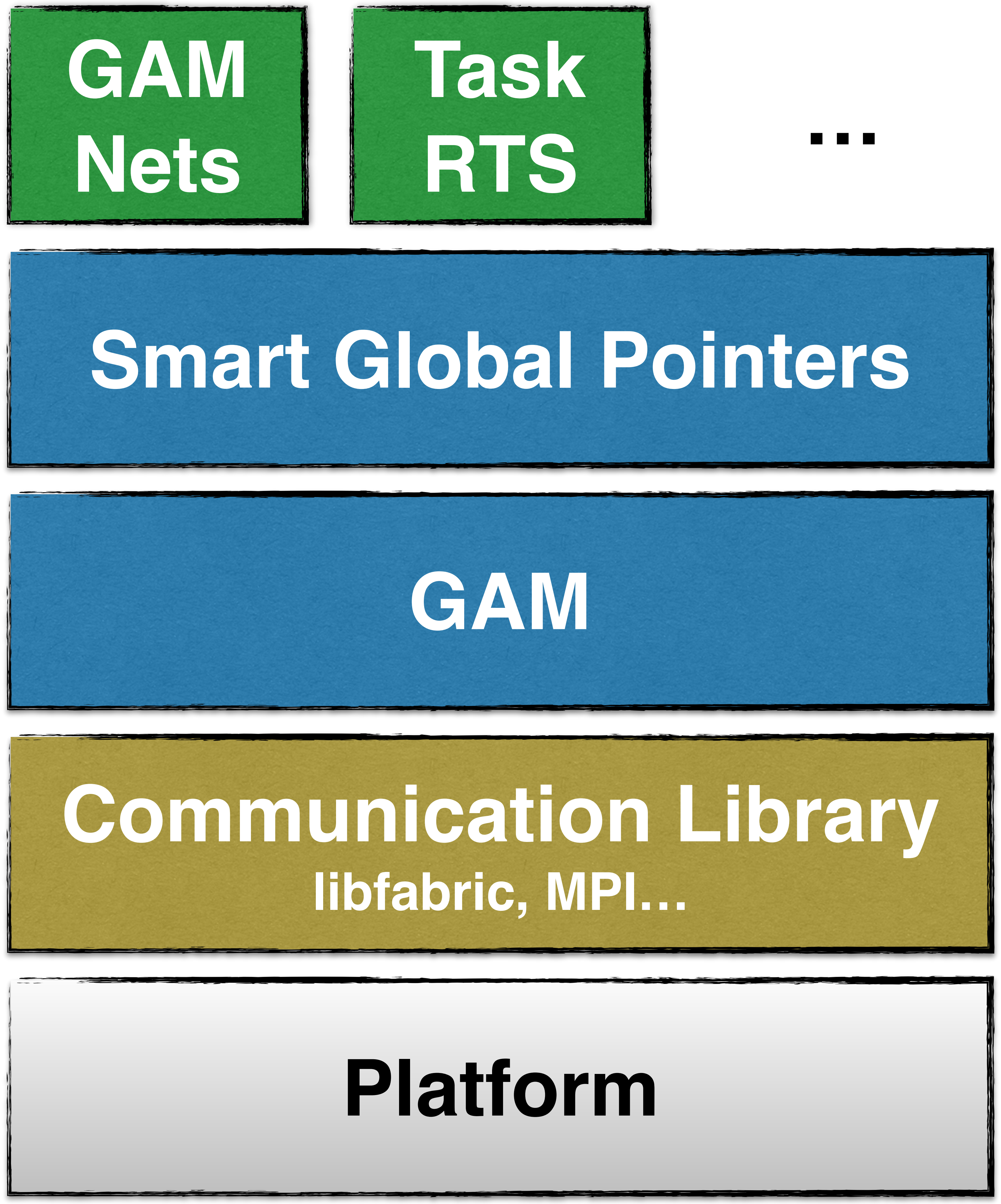
Key-Partitioning (KP)



Window-Farming (WF)



A NEW PROPOSAL FOR A STREAM-ORIENTED PGAS
DISTRIBUTED FASTFLOW V.2 — C++ GLOBAL MEMORY STACK



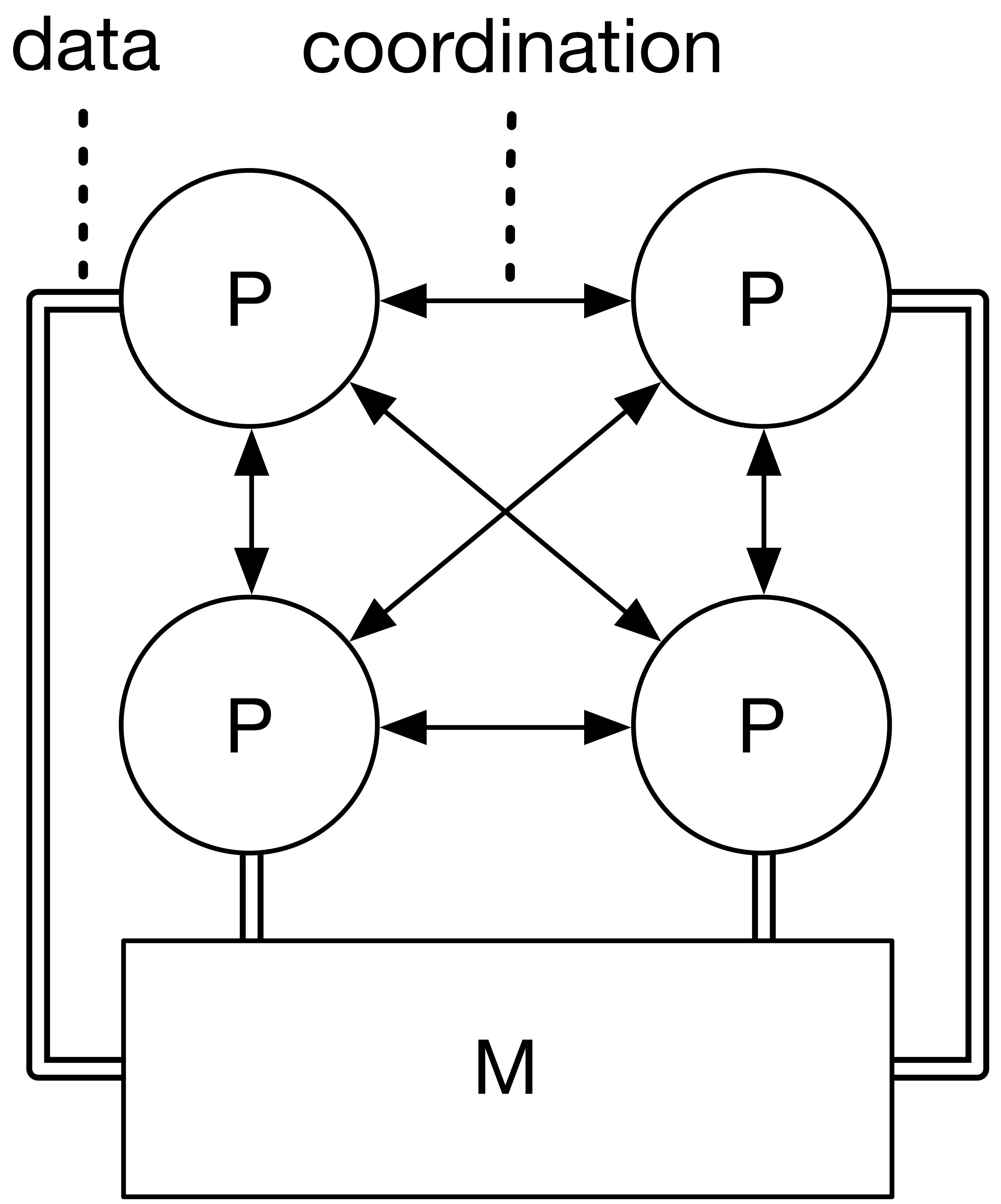
M. Drocco. Parallel Programming with Global Asynchronous Memory: Models, C++ APIs and Implementations. PhD. Thesis proposal, University of Torino, 2017 (not yet defended).

Aldinucci, S. Campa, M. Danelutto, P. Kilpatrick, and M. Torquati, “Targeting Distributed Systems in FastFlow,” in *Euro-Par 2012 Workshops,*

Global Asynchronous Memory
A new proposal for a STREAM-oriented PGAS (based on FF)

- From MPI Style: communicate pointers (a.k.a., capabilities)
- From DSM Style: shared address space

➡ Capability = both **data reference** and **synchronization token**



Public Capabilities

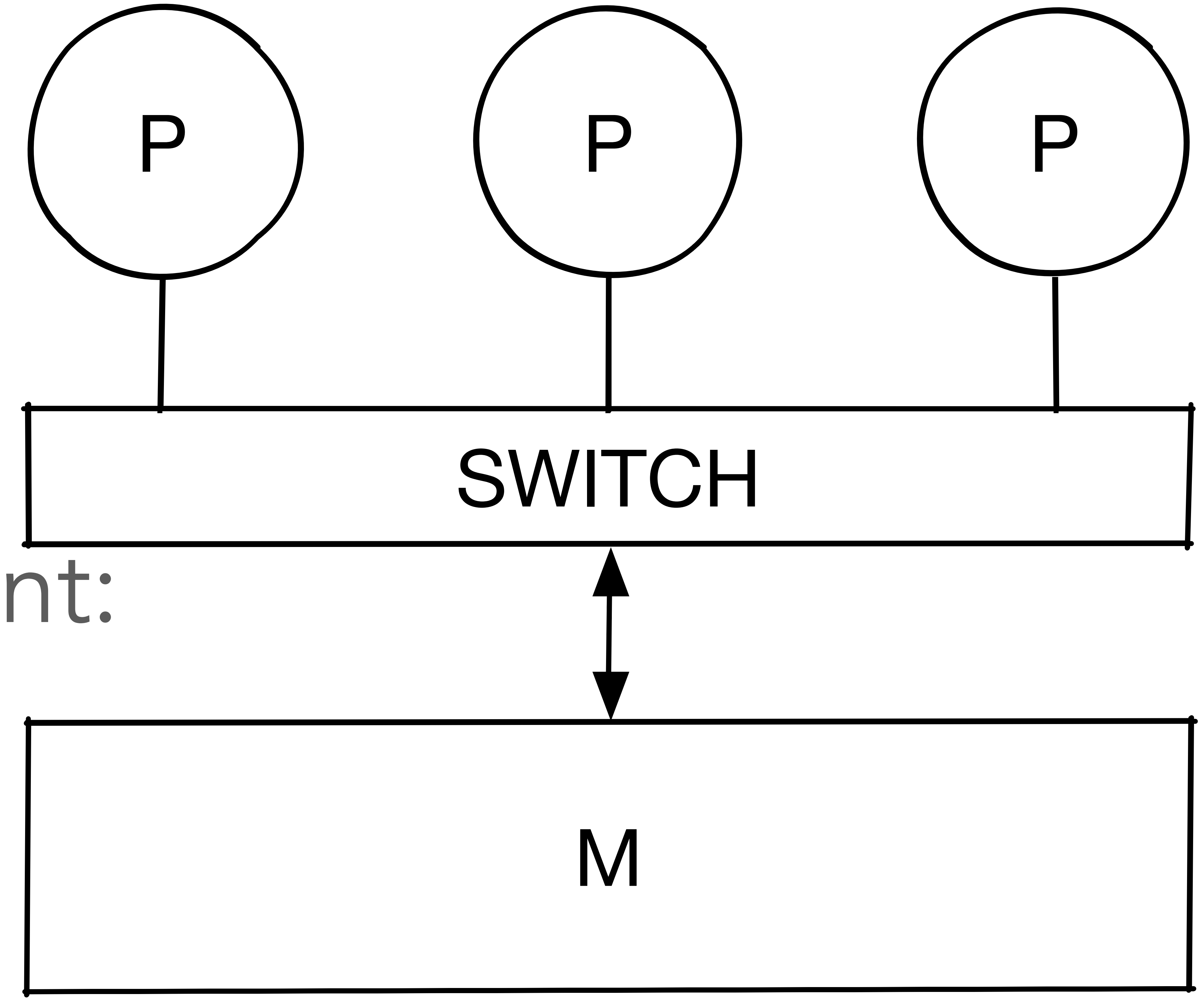
- Read-only (single assignment)
- Cacheable
- Can be copied

Private Capabilities

- Exclusive read-write
- Not cacheable
- Can be moved

GAM Memory Model

- (Trivial) Sequential Consistency
- **Avoiding** consistency issues (vs **solving** as in DSM/PGAS)
- SWMR cache-coherence invariant:
  - public → (NW)MR
  - private → SWSR



SMART GLOBAL POINTERS

- Rooted in modern C++
  - Intentional programming:
    public → shared, private → unique
- Automatic Memory Management — the C++ way
  - Smartness = memory-reference lifetime binding
  - No memory leaks, no dangling pointers
  - No garbage collection (vs Java & friends)

Public Pointers

```
public_ptr(T * const, Deleter);
public_ptr<T> make_public(Args&&…);

//copy constructor/assignment...
//move constructor/assignment...

public_ptr(private_ptr<T> &&);
public_ptr& operator=(private_ptr<T> &&);

std::shared_ptr<T> local();

void push(executor_id to);
public_ptr<T> pull_public(const exec_id from);
public_ptr<T> pull_public();
```

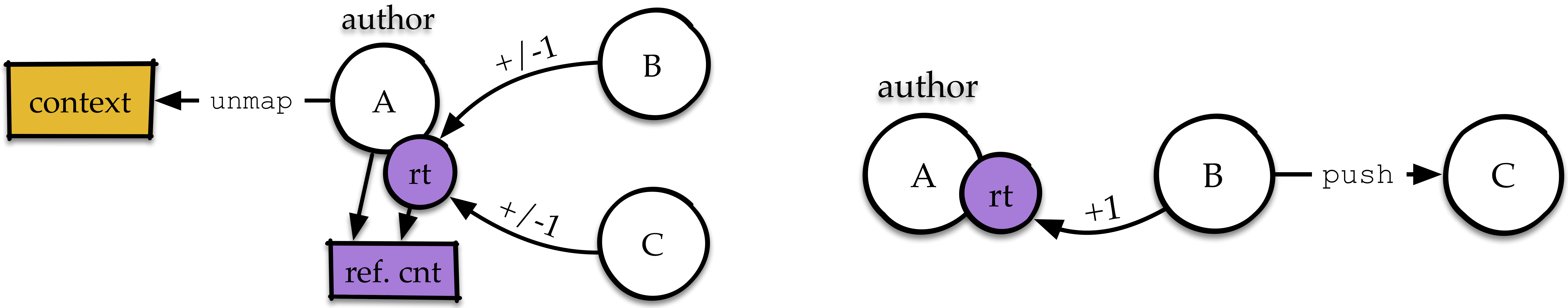
enables
plain C++ code

Private Pointers

```
private_ptr(T * const, Deleter);
private_ptr<T> make_private(Args&&…);

//move constructor/assignment…

//NO copy constructor/assignment

gam_unique_ptr<T> local(); //unique_ptr + custom deleter

void push(executor_id to);
private_ptr<T> pull_private(const exec_id from);
private_ptr<T> pull_private();
```

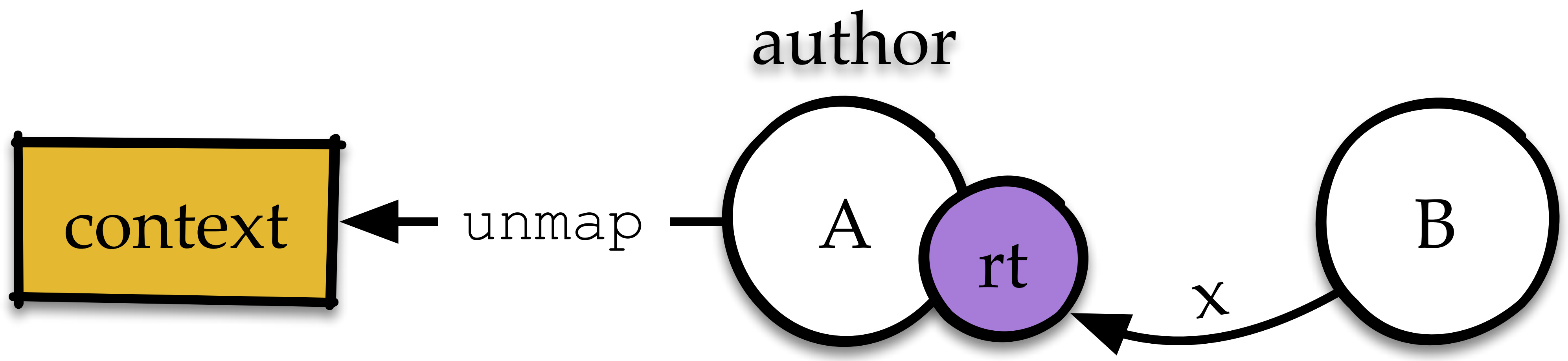
enables
plain C++ code

Smartness for Public Pointers



- Distributed **reference counting** protocol
- Creation/copy/push trigger +1, destruction triggers -1
- C++ shared pointers: atomic-based reference counting

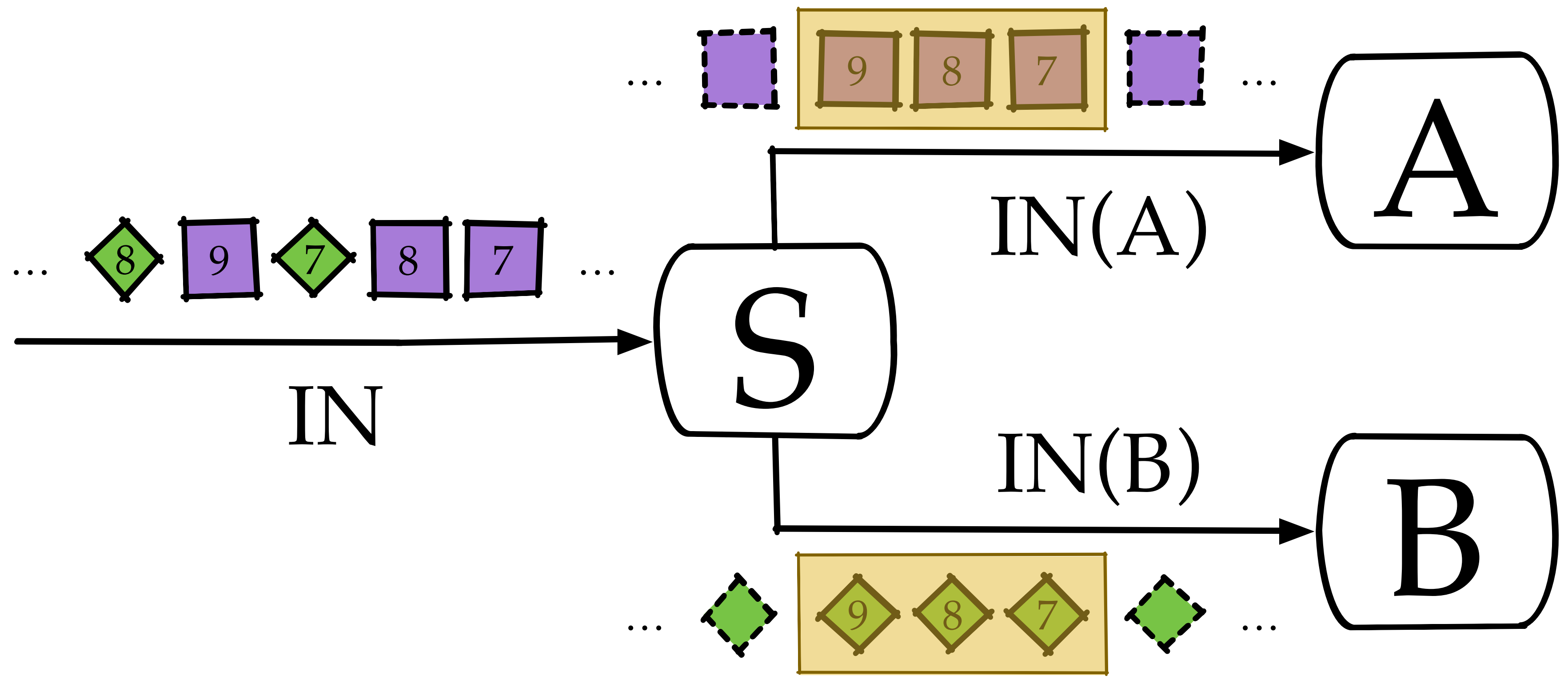
Smartness for Private Pointers



- Distributed **memory releasing** protocol
- Destruction triggers releasing
- C++ unique pointers: destruction-triggered release
- Inherently simpler than public pointers (as unique vs shared)

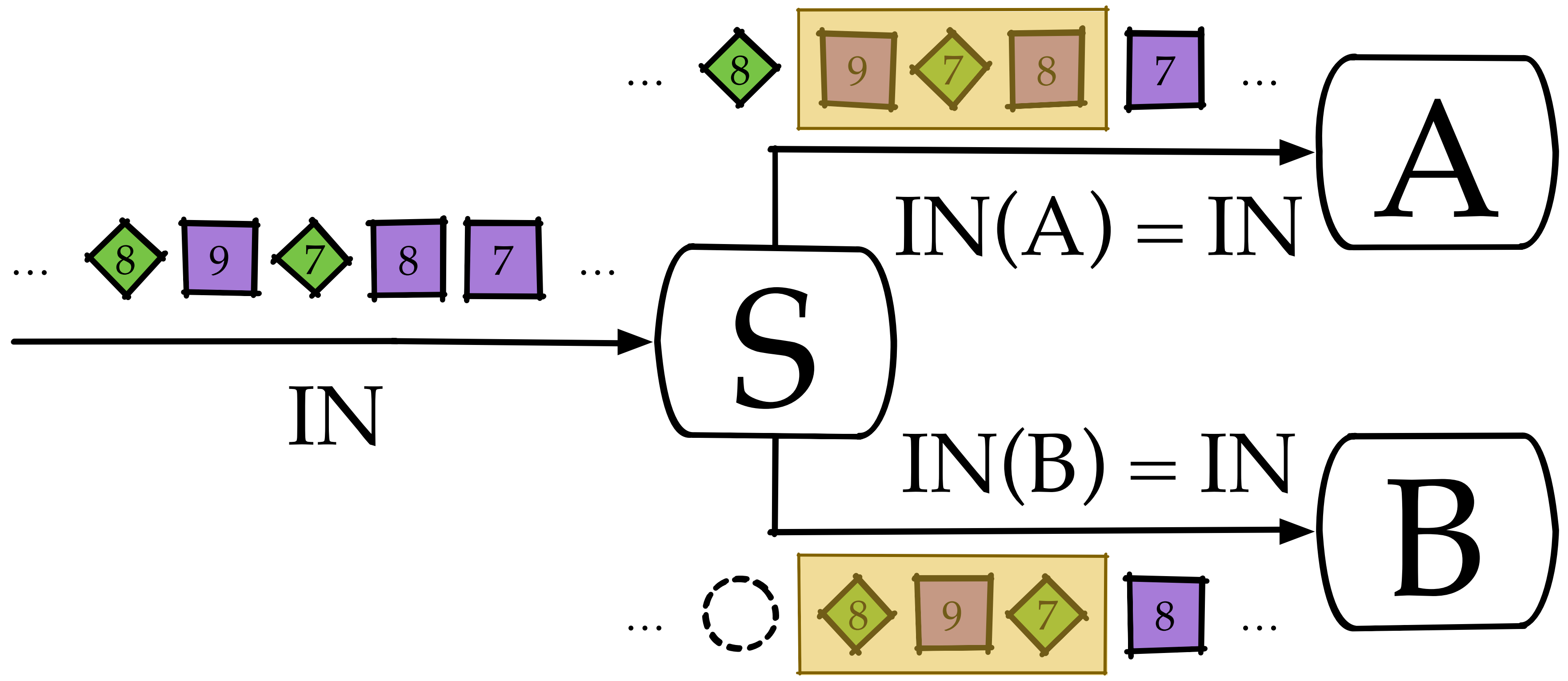
KP - INHERENTLY EFFICIENT

- Disjoint IN(A)/IN(B)
  - each tuple accessed "exclusively"
- GAM implementation
  - private pointers - exclusive capabilities
  - 1 RMA access per tuple



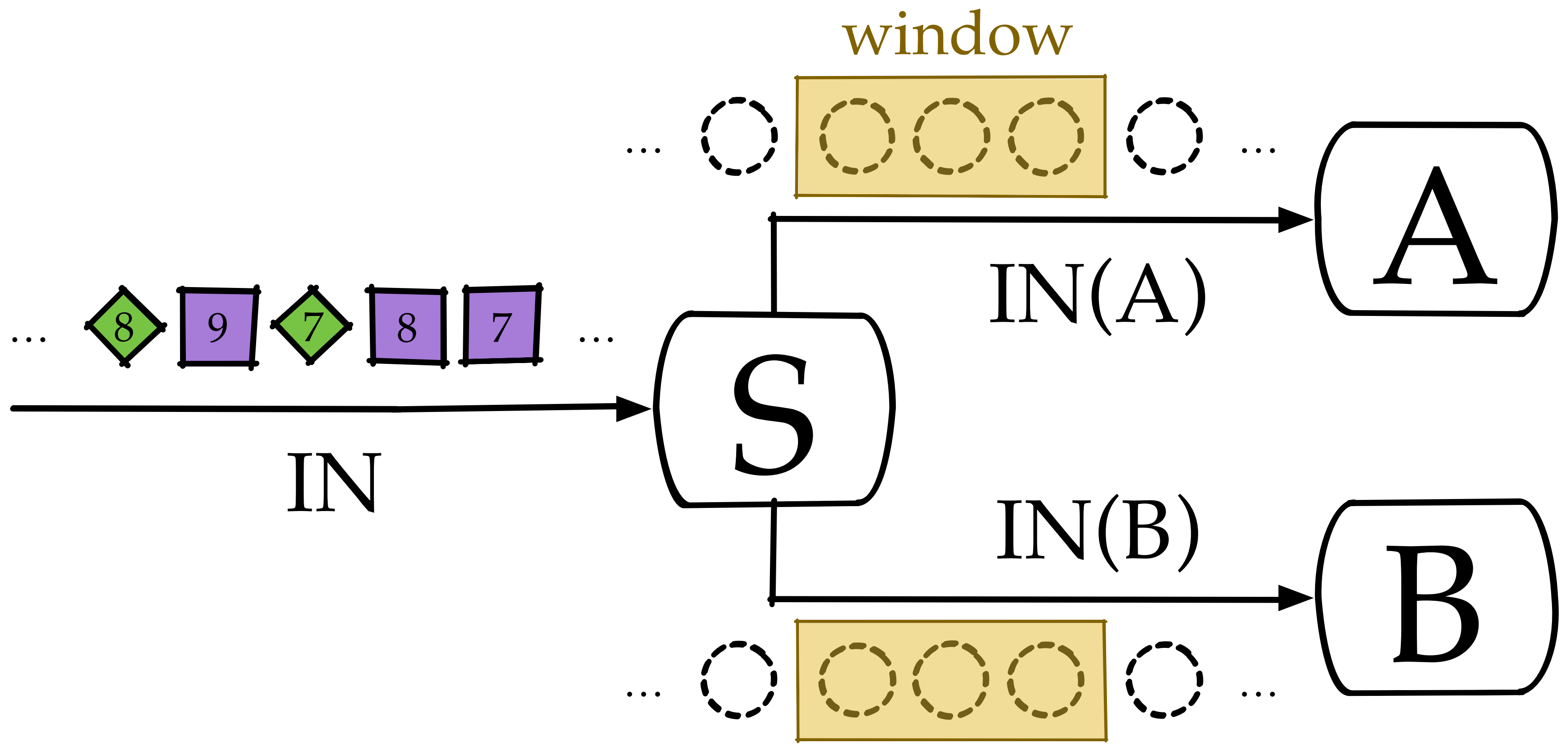
WF - Inherently Complex

- IN(A) = IN(B) = IN
  - each tuple accessed “by-any”
- GAM implementation
  - public pointers - read-only replicas
  - multiple RMA accesses per tuple

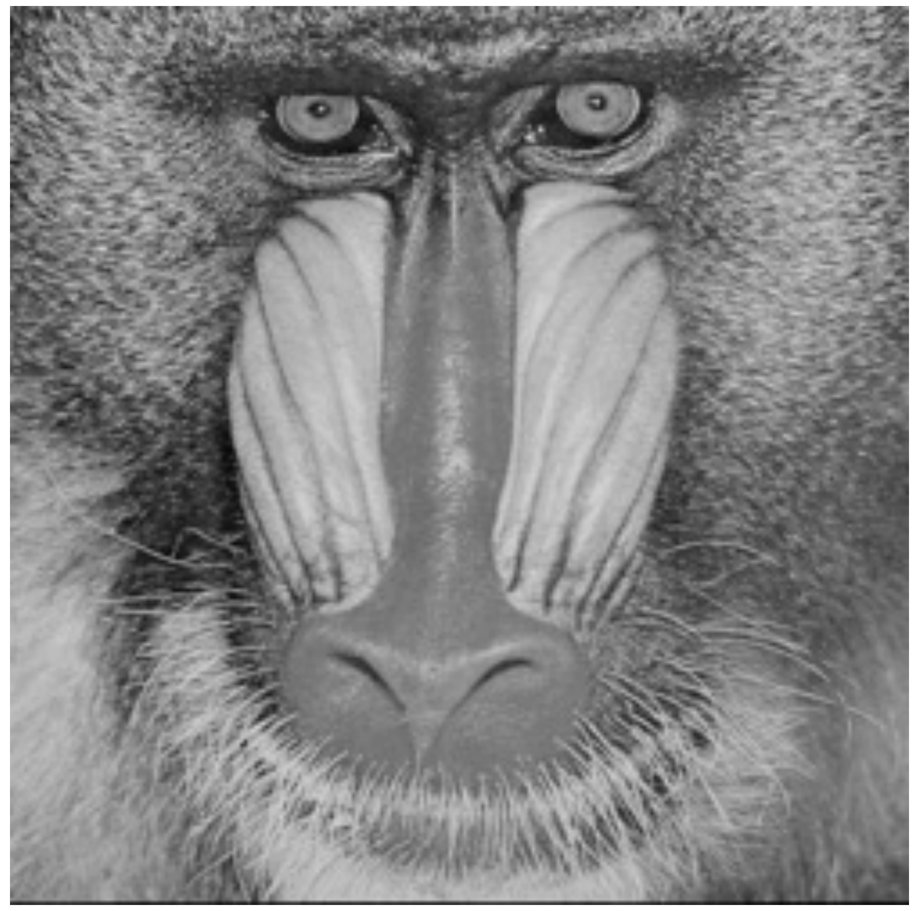


GAM Advantages

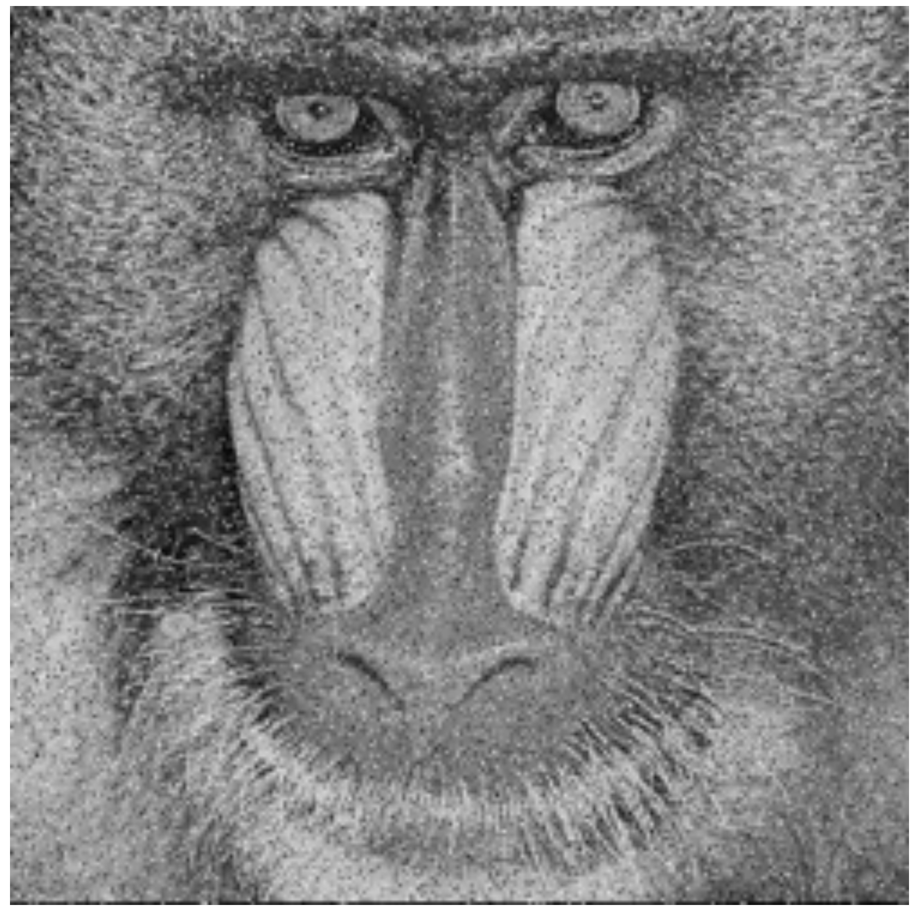
- Passing capabilities versus data
  - efficient worker-side windowing
  - extreme case: static dispatching not viable



HTP Video Restoration

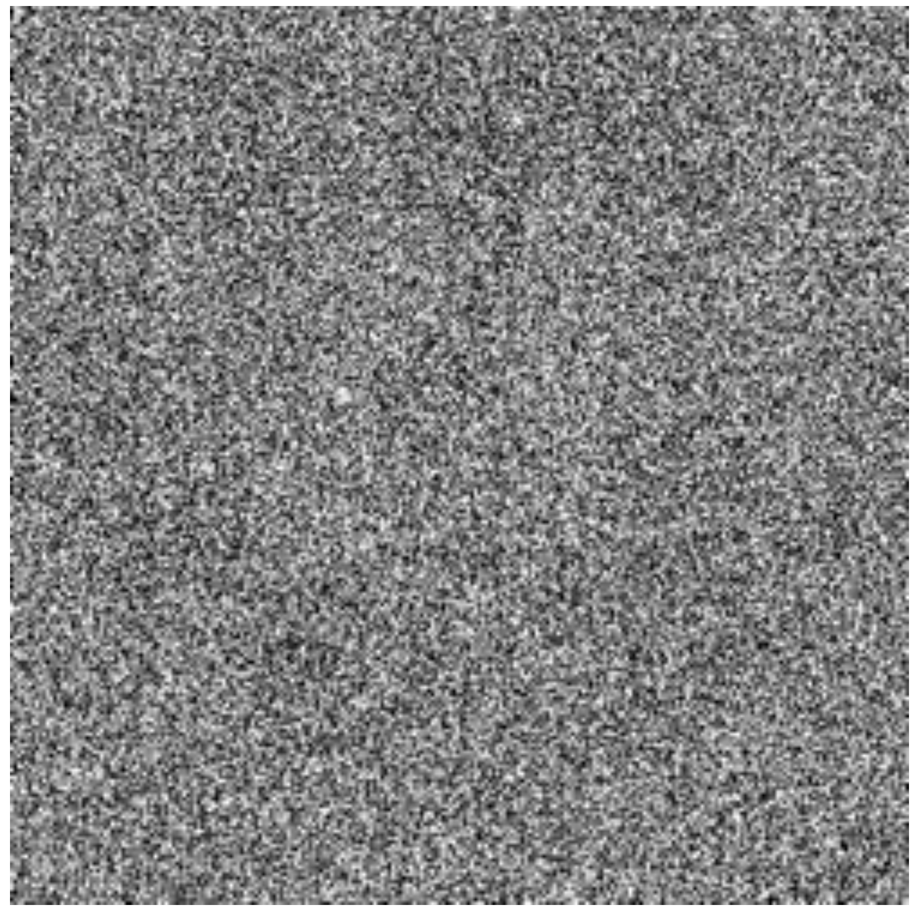


Original Baboon
1024x1024



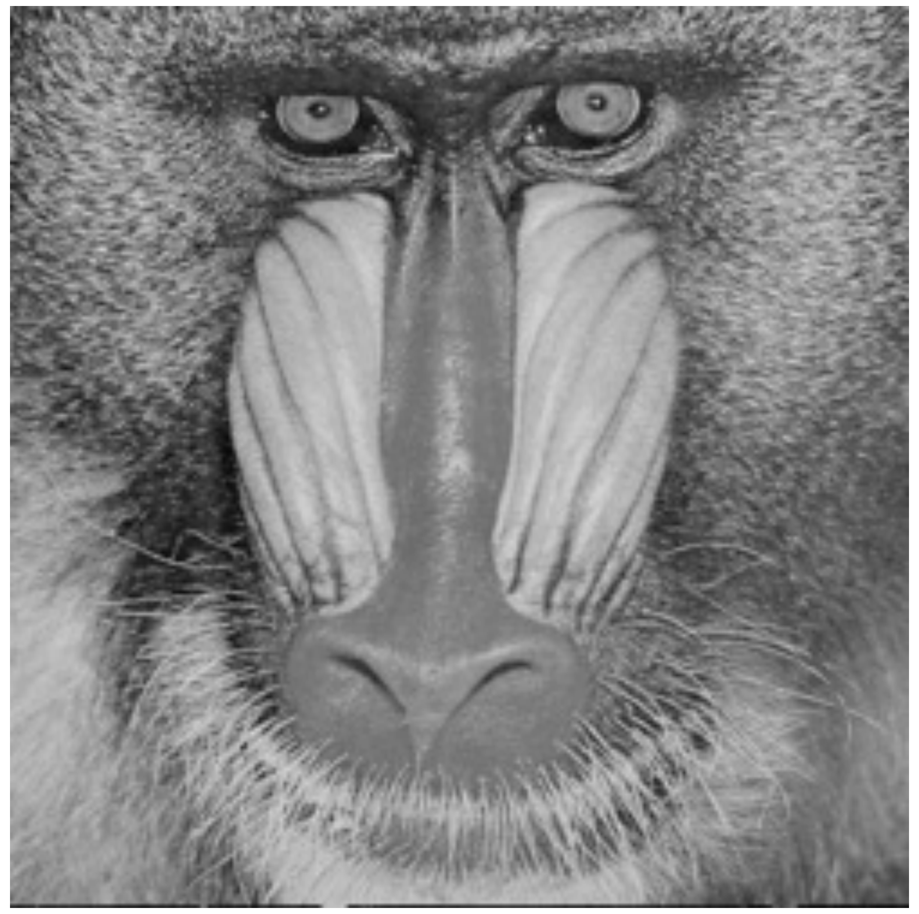
10% impulsive noise

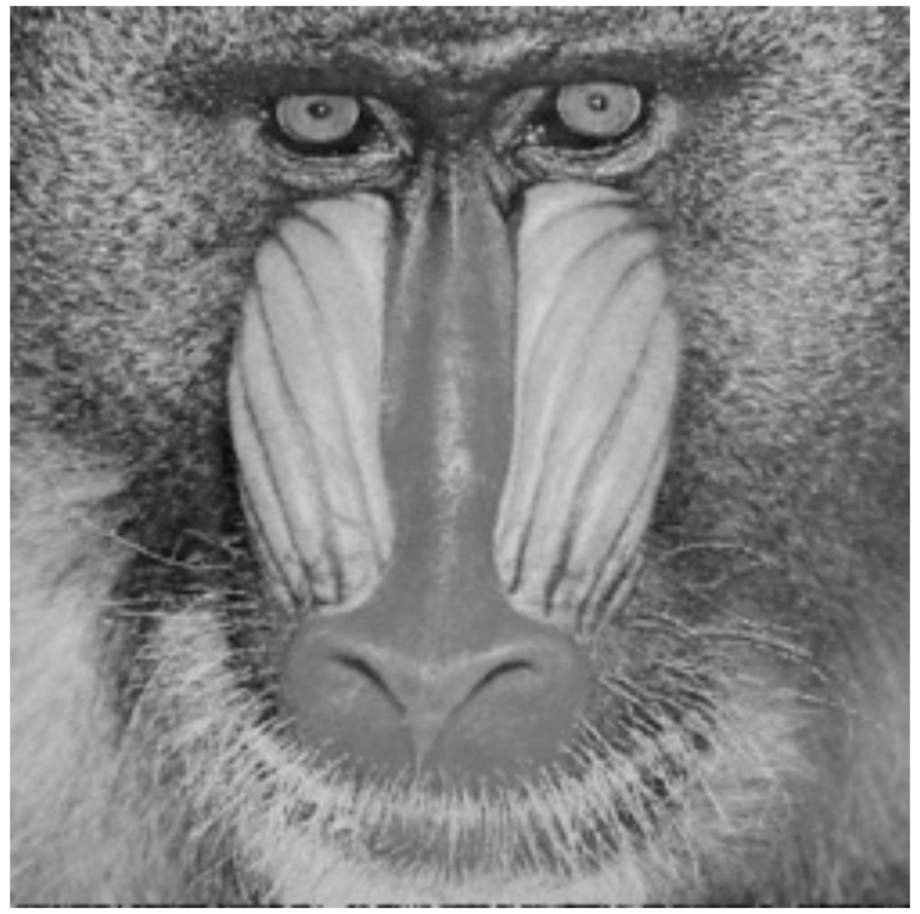
50% impulsive noise



90% impulsive noise

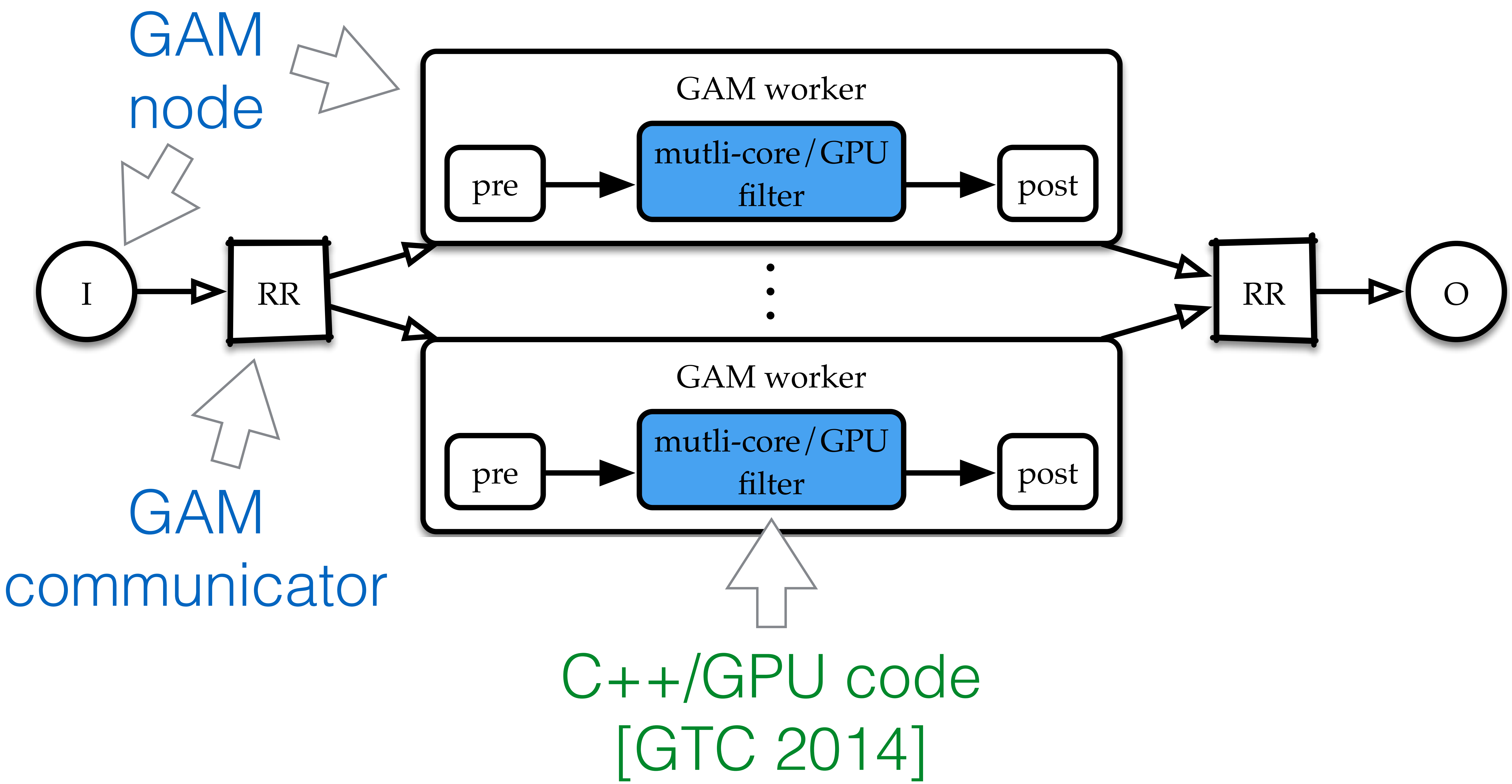
Restored



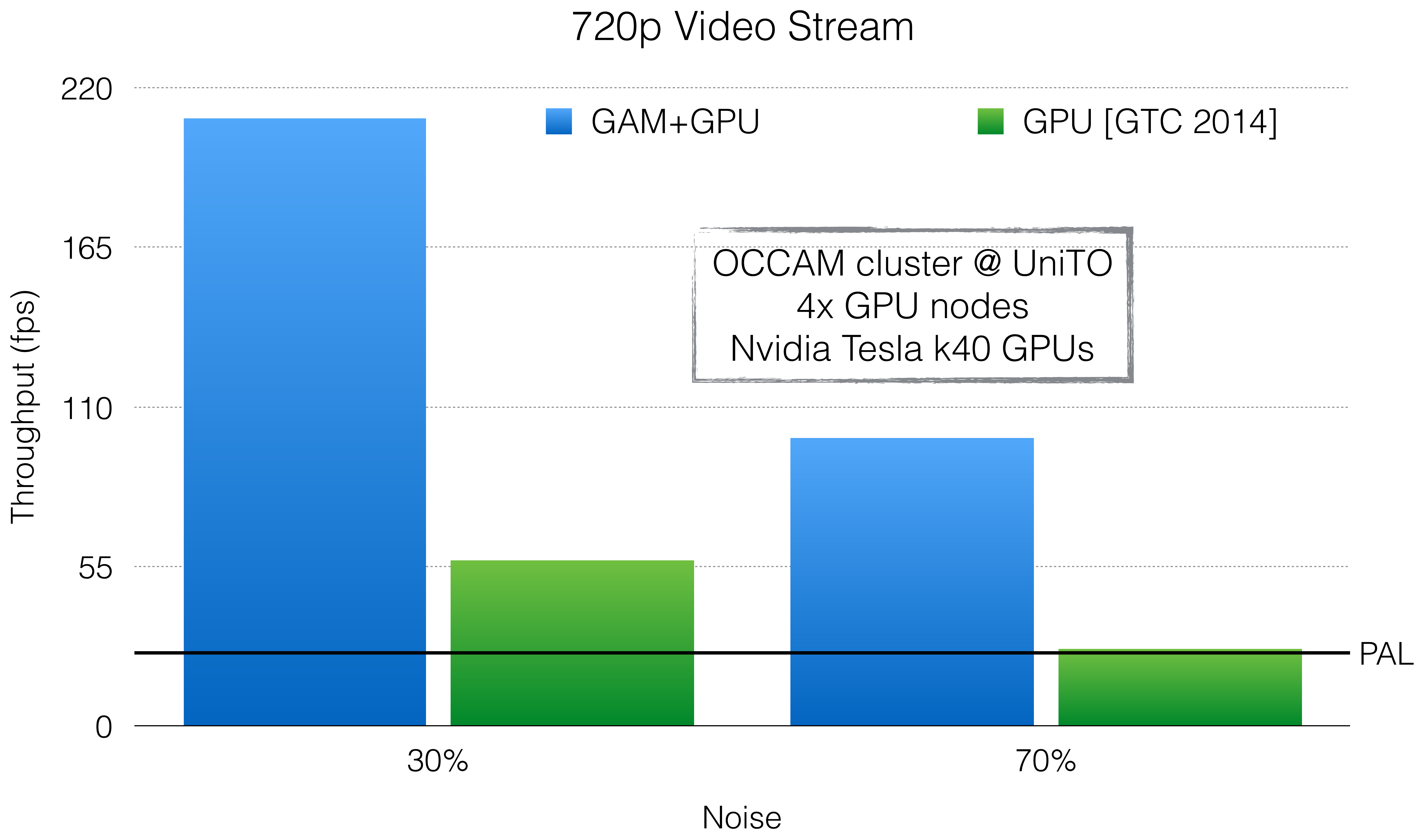


HTP Video Restoration

- A GAM implementation of **ordering farm**



HTP Video Restoration



EuroPar 2018

Torino, Italy — 27-31 August 2018

Co-chairs: M. Aldinucci, L. Padovani, M. Torquati