

Topic 4d – Memory Semantics and Codelet Execution Model



CPEG421/621 – Compiler Design

Outline

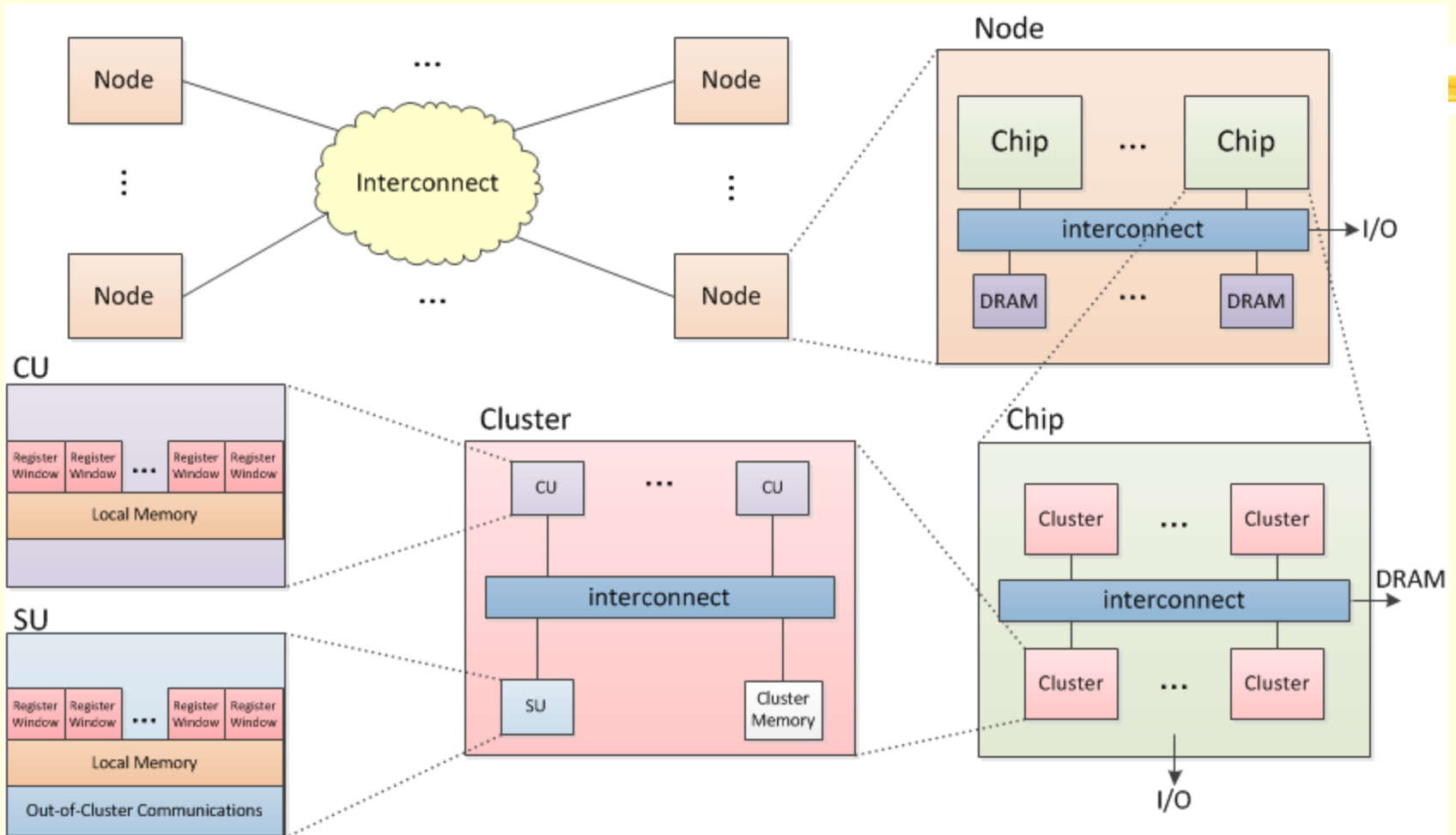


Introduction

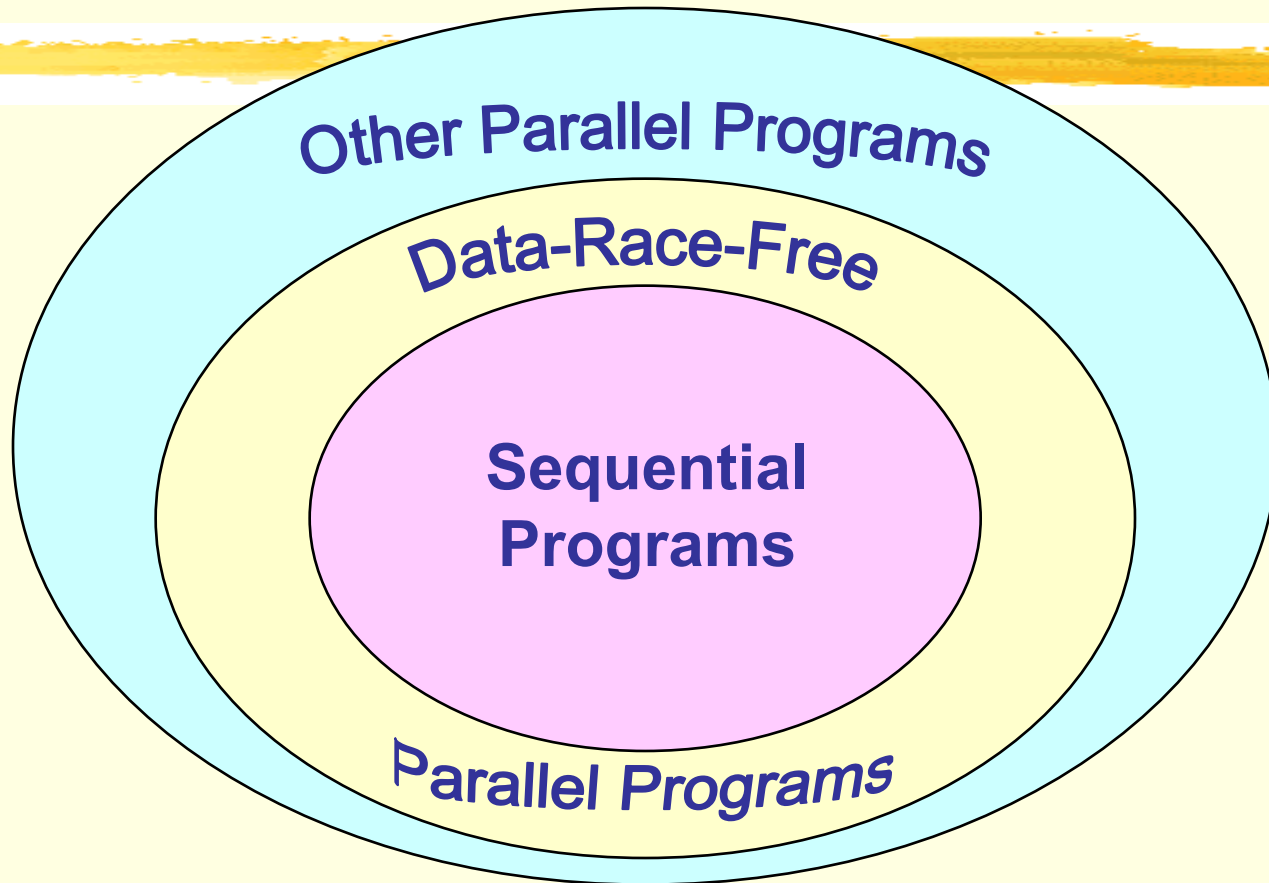
- **Memory Semantics: Three Key Questions**
 - Question Q1 and Location Consistency
 - Question Q2 and Q3
- **Memory Semantics and Codelet Execution Model**

Summary

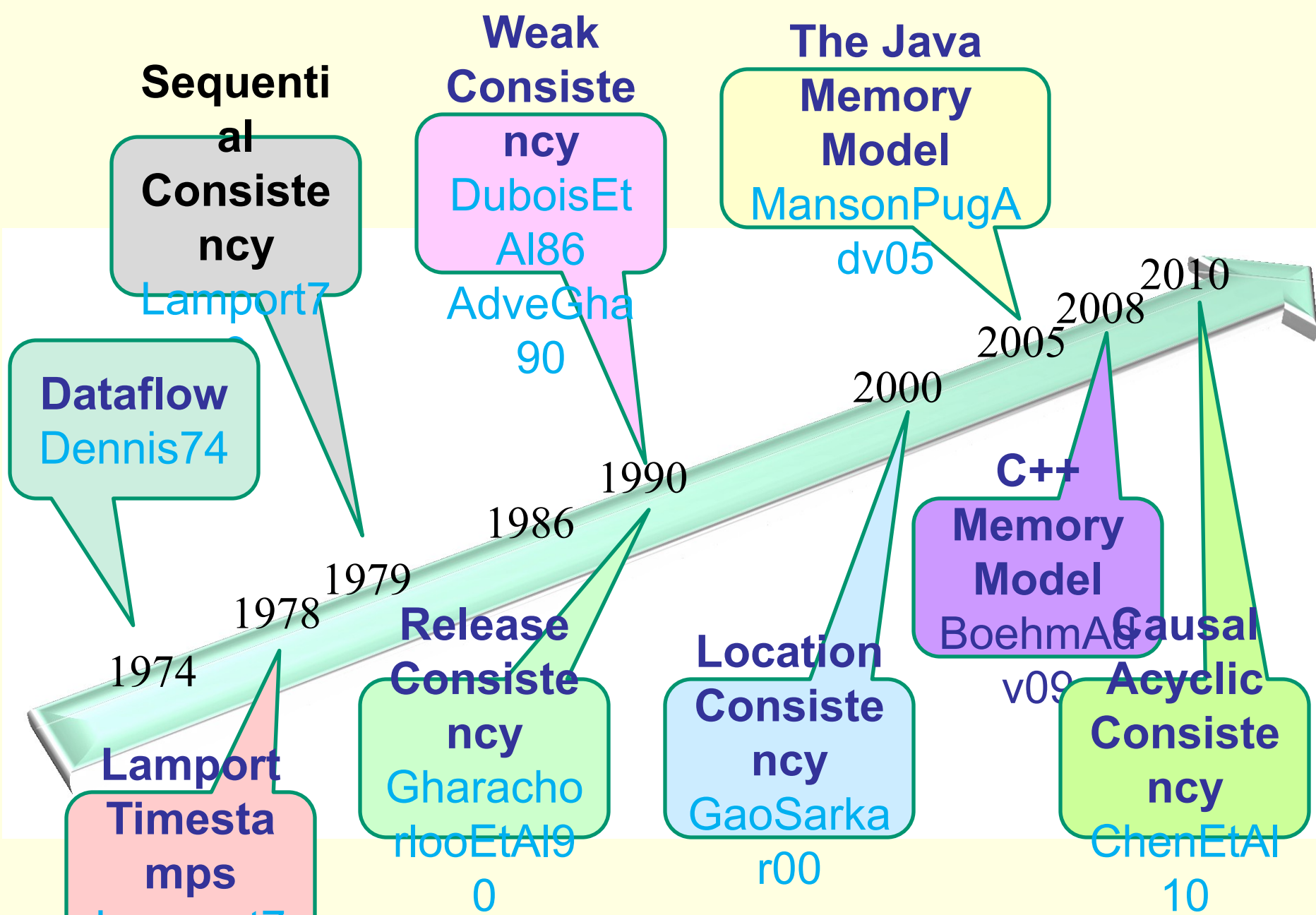
An Abstract Machine Model



Some Philosophical Remarks



- 1) Make **common case efficient** (default = no coherence)
- 2) If you need coherence, **SAY SO**.

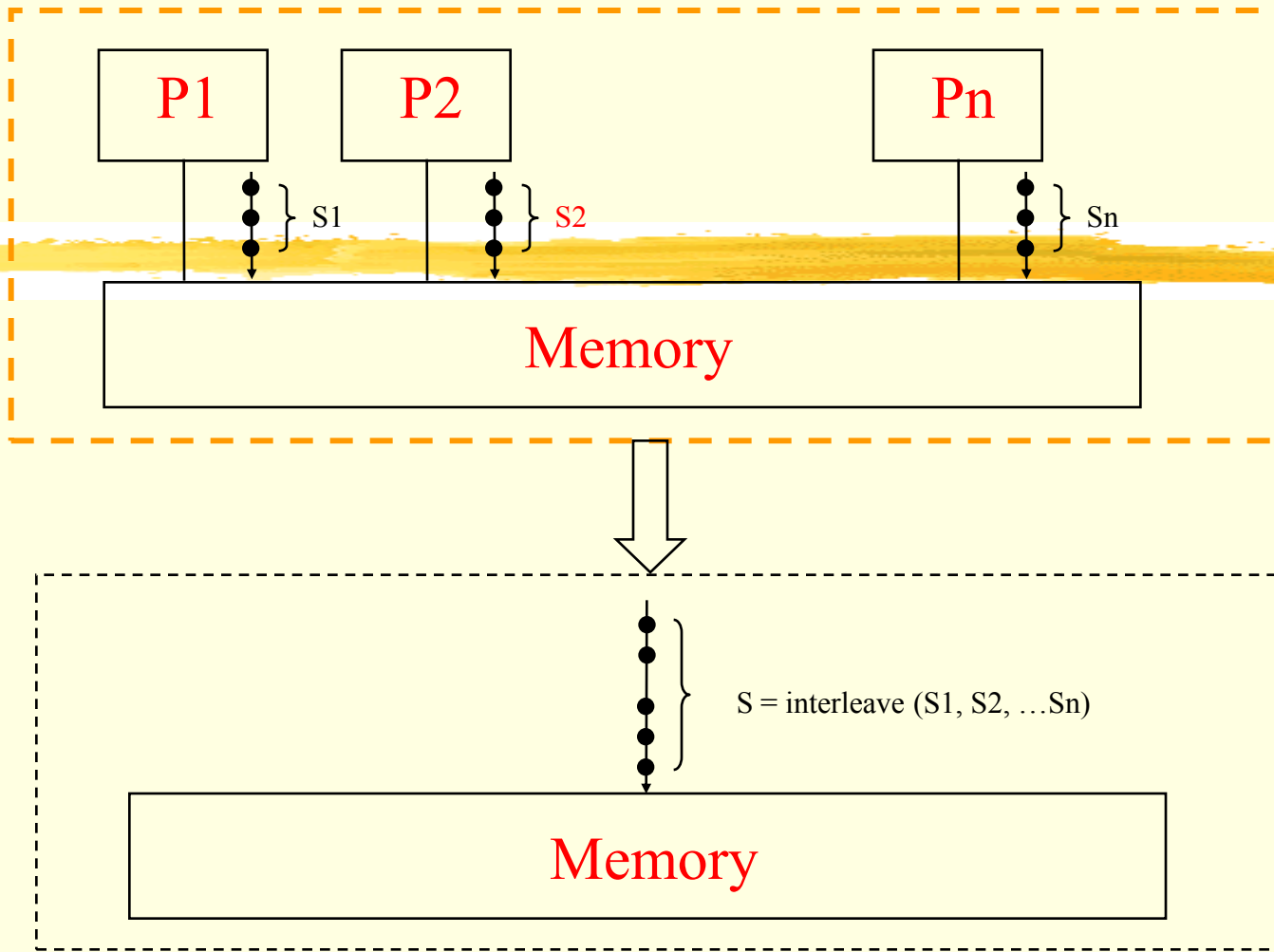


Timeline of Memory Models

Sequential Consistency (SC)

[Hardware is *sequentially consistent* if] the result of **any execution** is the same *as if* the operations of *all the processors* were executed in *some sequential order*, and the operations of each individual processor appear in this sequence in the order specified by its program.

[Lamport 79]



The SC Memory Model

“Memory Coherence”

A Basic Assumption of SC-Derived Memory Models



“...All writes to the same location are serialized in some order and are performed in that order with respect to any processor...”

[Gharacharloo Et Al 90]

Open Questions



- Is the SC model *easier* for programmers ?
- Is the performance gain due to *relaxed SC-derived* model worth the *complexity* ? ([Hill'98])

Outline



- **Introduction**
- **Memory Semantics: Three Key Questions**
 - Question Q1 and Location Consistency
 - Question Q2 and Q3
- **Memory Semantics and Codelet Execution Model**
- **Summary**

Three Key Question on Memory Models

Q1: What happens when two (or more) concurrent load/store operations happen (arrives) at the same memory location?

Answers ?

Another Two Key Questions



Assuming two memory operations with the same destination memory location address X (i.e. LOAD X or STORE X) are issued through the same processing core. Should a memory model allow them to become *out-of-order* along the way ?

Outline



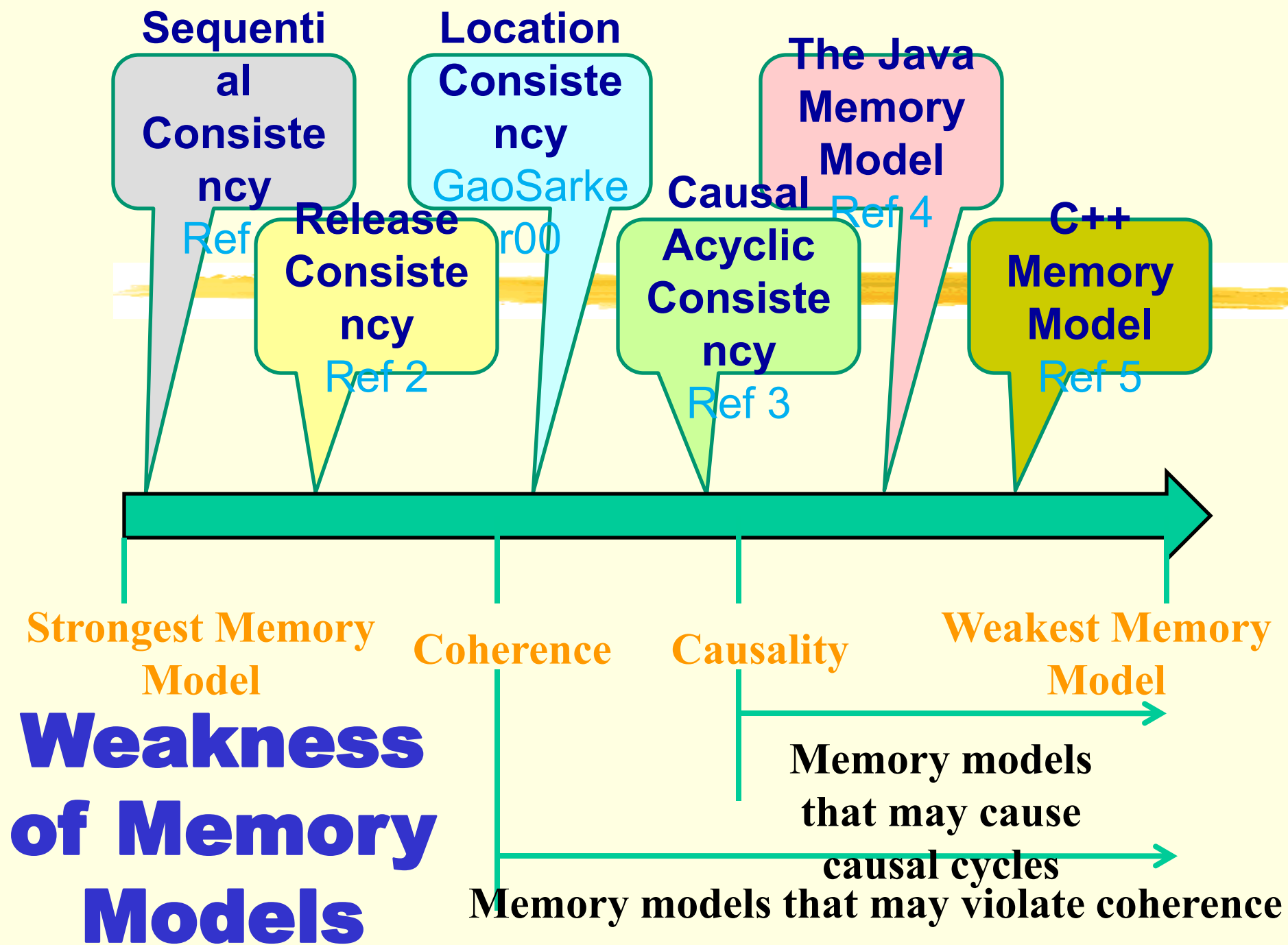
- **Introduction**
- **Memory Semantics: Three Key Questions**
 - **Question Q1 and Location Consistency**
 - **Question Q2 and Q3**
- **Memory Semantics and Codelet Execution Model**
- **Summary**

Question Q1 on Memory Models

Q1: What happens when two (or more) concurrent load/store operations happen (arrives) at the same memory location?

Answers ?

- Dataflow models (e.g. I-structure/M-Structure) ?
- Sequential consistency (SC) ?
- Release Consistency (RC) model ?
- Java JMM ?
- C++ thread model ?
- Location Consistency (LC) ?
- Others ?






**Question: Can we Remove the
“Memory Coherence”
barriers?**

Answer: Yes!

**By intuition, The answer is
“Yes”!**



That is:

**If you need an order to be “enforced”
between two memory ops by hardware –
Say it!**

**Otherwise, hardware should not have an
obligation to “serialize” the memory
operation!**

Thread1
 $w_1: L := val_1$
...
 $w_2: L := val_2$
...

An Example

Thread2
 $w_3: L := val_3$
...
...

...
 $r_1: read\ L$
 $sync(t_1, t_2)$

...
 $r_2: read\ L$
 $sync(t_1, t_2)$
 $r_3: read\ L$



States of L:

$2(t_1, val_1)$

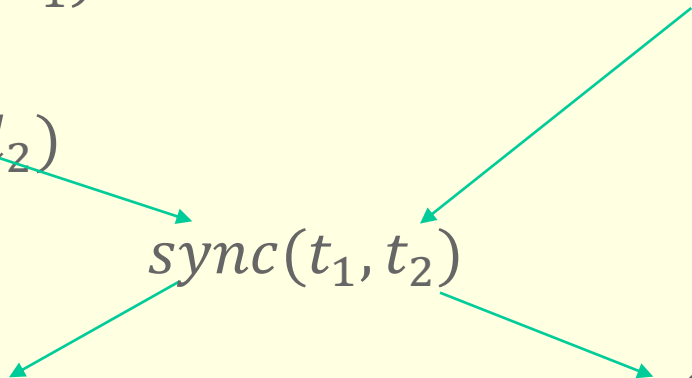
(a “growing” pomset!)

$w(t_2, val_3)$

↓
 $w(t_1, val_2)$

$sync(t_1, t_2)$

$r(t_2)$



Outline



- Introduction
- Memory Semantics: Three Key Questions
 - Question Q1 and Location Consistency
 - **Question Q2 and Q3**
- Memory Semantics and Codelet Execution Model
- **Summary**

Another Two Key Questions Related to Memory Models



Assuming two memory operations with the same destination memory location address X (i.e. LOAD X or STORE X) are issued through the same processing core.

Notes: We assume that the two memory operations are issued in their program order. Both of the two memory operations access memory location address X .

Q2: Should the hardware (architecture) permit > 1 alternative paths of routing of the memory operations (transactions) along the way?

Q3: If the answer of Q2 is true (I assume it is) - then it is well possible that the two operations may arrive at its destination out-of-order?

Your Answers to the Questions ?

Q1: Should the hardware (architecture) permit > 1 alternative paths of routing of the memory operations (transactions) along the way?

Q2: If the answer of Q1 is true (I assume it is) - then it is well possible that the two operations may arrive at its destination out-of-order?

No.	Answer to Q1	Answer to Q2	Which one ?
1	Yes	Yes	
2	Yes	No	
3	No	Yes	
4	No	No	

Possible Answers to the Questions

- Q1:** Should the hardware (architecture) permit > 1 alternative paths of routing of the memory operations (transactions) along the way?
- Q2:** If the answer of Q1 is true (I assume it is) - then it is well possible that the two operations may arrive at its destination out-of-order?

No.	Answer to Q1	Answer to Q2	Who answered
1	Yes	Yes	GG, MM
2	Yes	No	BS
3	No	Yes	MS
4	No	No	DD,SS

Outline



- **Introduction**
- **Memory Semantics: Three Key Questions**
 - **Question Q1 and Location Consistency**
 - **Question Q2 and Q3**
- **Memory Semantics and Codelet Execution Model**
- **Summary**

Memory Model of Codelets



The shared memory model is based on LC (Location Consistency, Gao and Sarkar 2000) and its variants/extensions.

There is *no global coherence requirement* due to LC

Our answer to the 3 questions (Q0, Q1, Q2) will lead to extensions to LC: *Work In Progress!*

Outline




- **Introduction**
- **Memory Semantics: Three Key Questions**
 - **Question Q1 and Location Consistency**
 - **Question Q2 and Q3**
- **Memory Semantics and Codelet Execution Model**
- **Summary**

Summary



- The **memory model** of a **PXM** will help define its **scalability**
- Classical PXMs (SC,...) do NOT scale well on future manycore architectures
- We want to get rid of coherence, without throwing causality away.
- Relaxed memory models, such as Location Consistency, can help design bigger, more scalable manycore systems.
- Programming languages need to be aware of parallelism
 - Need to know about the underlying memory model
 - Even traditionally sequential languages (C,C++) are starting to provide a crude memory model to handle concurrency and parallelism
 - Other languages, designed to be “concurrency-aware” (Java, X10, Chapel, ...) provide a more refined memory model – but maybe still too relaxed



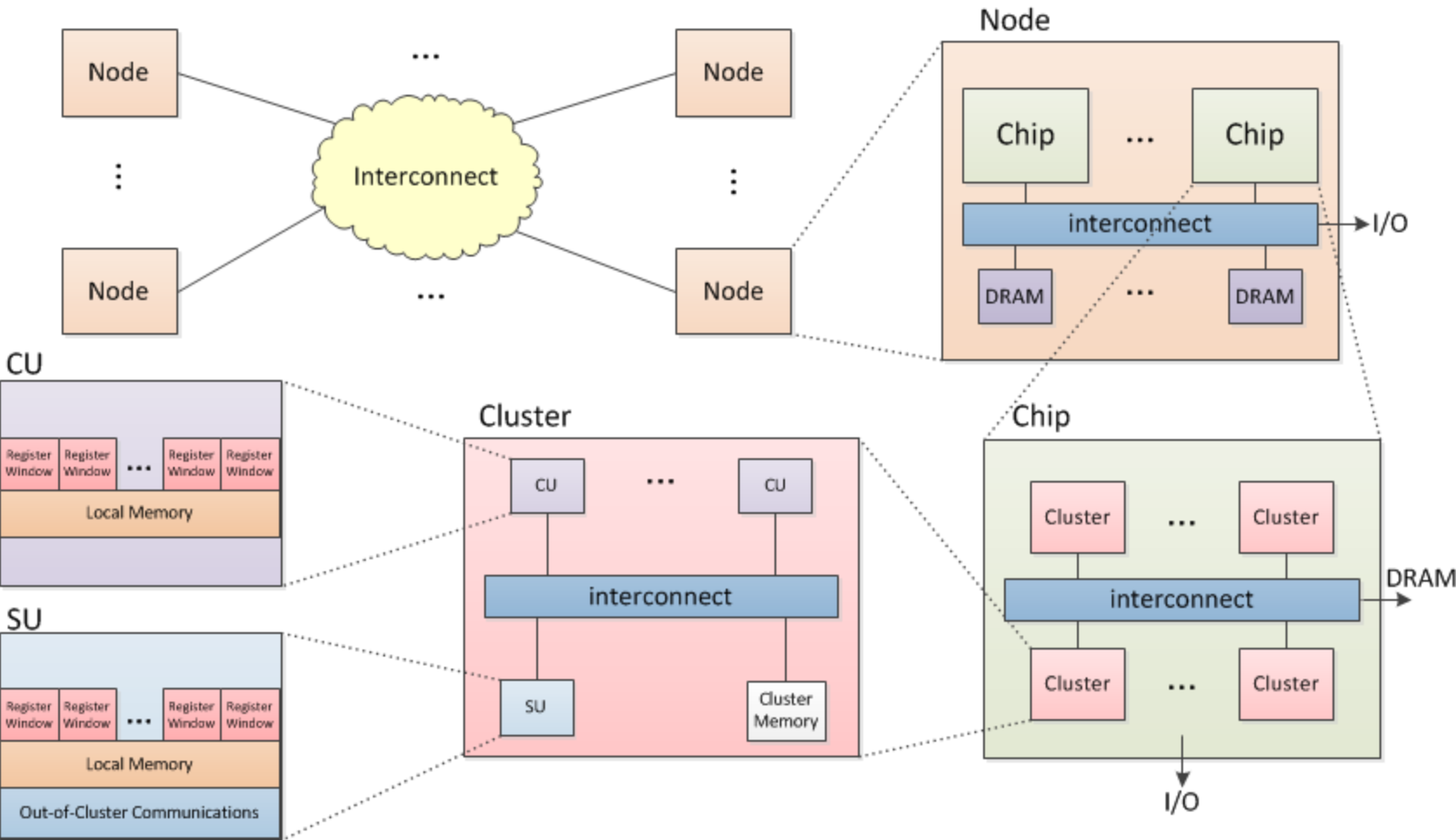
Topic 4e – Using the Codelet Model for Exascale Computations

Introduction: Exploiting Parallelism in Many-Core Architectures



- Many-core chips are finally here
- Current control-flow based frameworks (MPI, OpenMP) incur too much overhead for exascale computing (coarse-grain parallelism)
- To efficiently exploit parallelism, fine-grain approaches should be preferred
- We propose a Codelet Program Execution Model, based on dataflow theory

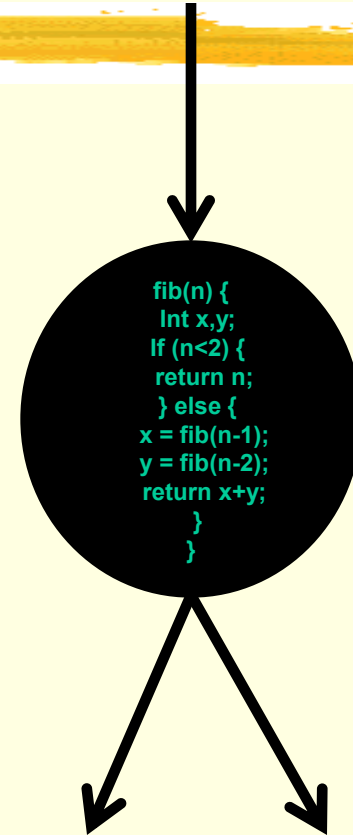
An Abstract Machine Model



The Codelet Model

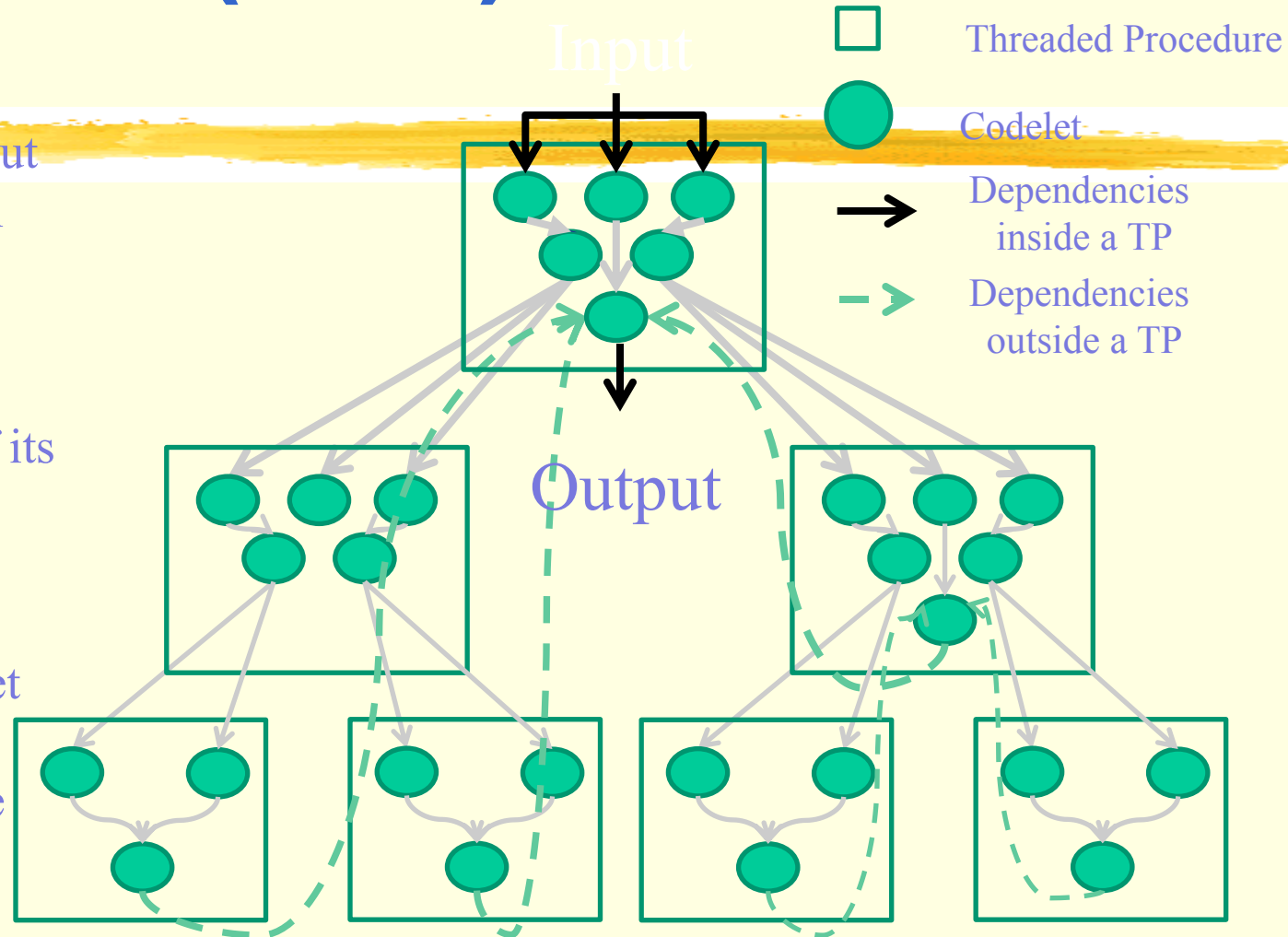
Goals: to effectively represent data and computing resource sharing through a hybrid dataflow approach, using two levels of parallelism

Definition: a codelet is a sequence of machine instructions which act as an atomically-scheduled unit of computation.



The Codelet Graph Model (CDG)

- A CDG is well-behaved if, when input tokens are present on all input arcs, it consumes all of its tokens and produces one token on each of its output arcs
- Well-behaved CDGs ensure *determinate* results: for a given set of input tokens corresponds a unique set of output tokens



Achieving Exascale Performance



- Loop parallelism and Codelet Pipelining: SWP applied to multi/many cores with SSP
- Sync-Back Continuations (SBC):
 - Evolution of “futures” and “continuations”
 - Long-latency operations
 - SBCs are asynchronous.
- Meeting Locality Requirements
 - Codelets inputs are supposed to be locally available.
 - A codelet can perform a SBC to retrieve the missing data.
 - Percolation can bring code and/or data preemptively to the codelet.

Smart Adaptation in an Exascale CXM: Power, Energy, and Resiliency

- Power Management & Energy Efficiency
 - Percolation: moving code and/or data efficiently where needed.
 - Self-Aware Power Management: the system decides of scheduling and power policies according to goals and dynamic events
- Achieving Resiliency on $10^5 - 10^6$ cores
 - Duplicating computation on various parts of the system
 - Actively looking for badly-behaving cores
 - Check-pointing (easily with CDGs)

Conclusion



- Codelets are fine-grain, atomically scheduled sequences of code, grouped into codelet graphs.
- The use of sync-back continuations and parallel loop SWP will enable codelets to make as many cores busy as possible
- Percolation can improve both data and code locality, as well as energy efficiency
- The codelet PXM bets on self-awareness to ensure reliability
- A runtime system inspired by codelets already exists (SWARM, by ETI).
- We are extending LLVM to be codelet-aware.