Multicore versus FPGA in the Acceleration of Discrete Molecular Dynamics*+

Tony Dean~ Josh Model# Martin Herbordt

Computer Architecture and Automated Design Laboratory
Department of Electrical and Computer Engineering
Boston University
http://www.bu.edu/caadlab

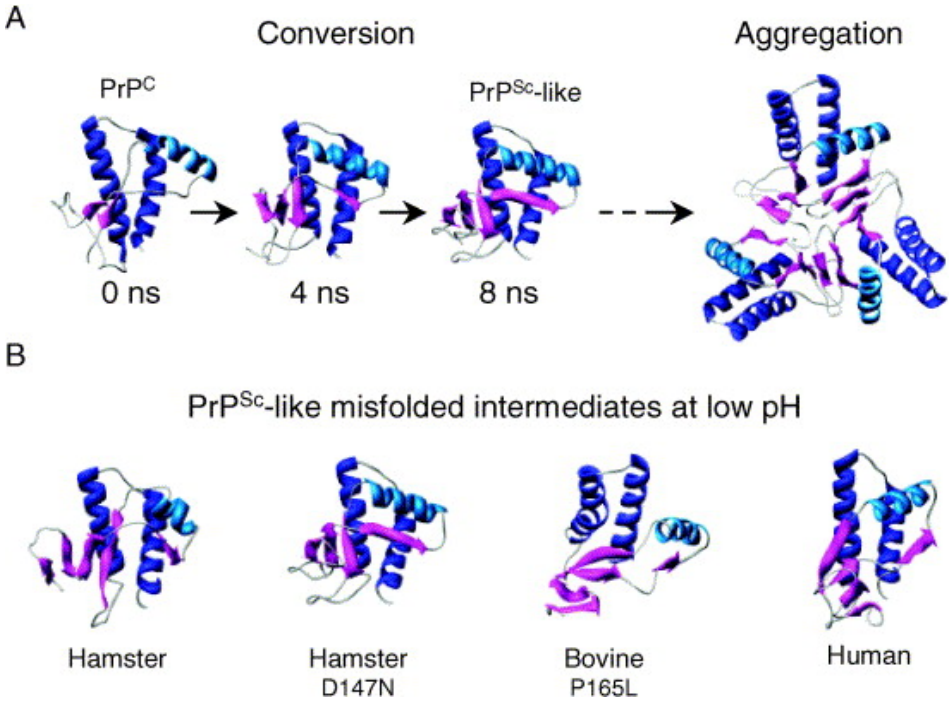
* This work supported, in part, by MIT Lincoln Lab and the U.S. NIH/NCRR
+ Thanks to Nikolay Dokholyan, Shantanu Sharma, Feng Ding, George Bishop, François Kosie
~ Now at General Dynamics
# Now at MIT Lincoln Lab

Overview – mini-talk

- **FPGAs are effective niche accelerators**
  - **especially suited for fine-grained parallelism**
- **Parallel Discrete Event Simulation (PDES) is often not scalable**
  - **need ultra-low latency communication**
- **Discrete Event Simulation of Molecular Dynamics (DMD) is**
  - **a canonical PDES problem**
  - **critical to computational biophysics/biochemistry**
  - **not previously shown to be scalable**
- **FPGAs can accelerate DMD by 100x**
  - **Configure FPGA into a superpipelined event processor with speculative execution**
- **Multicore DMD by applying FPGA method**

Why Molecular Dynamics Simulation is so important …

- Core of Computational Chemistry
- Central to Computational Biology, with applications to
  - → Drug design
  - → Understanding disease processes …



From DeMarco & Dagett: PNAS 2/24/04

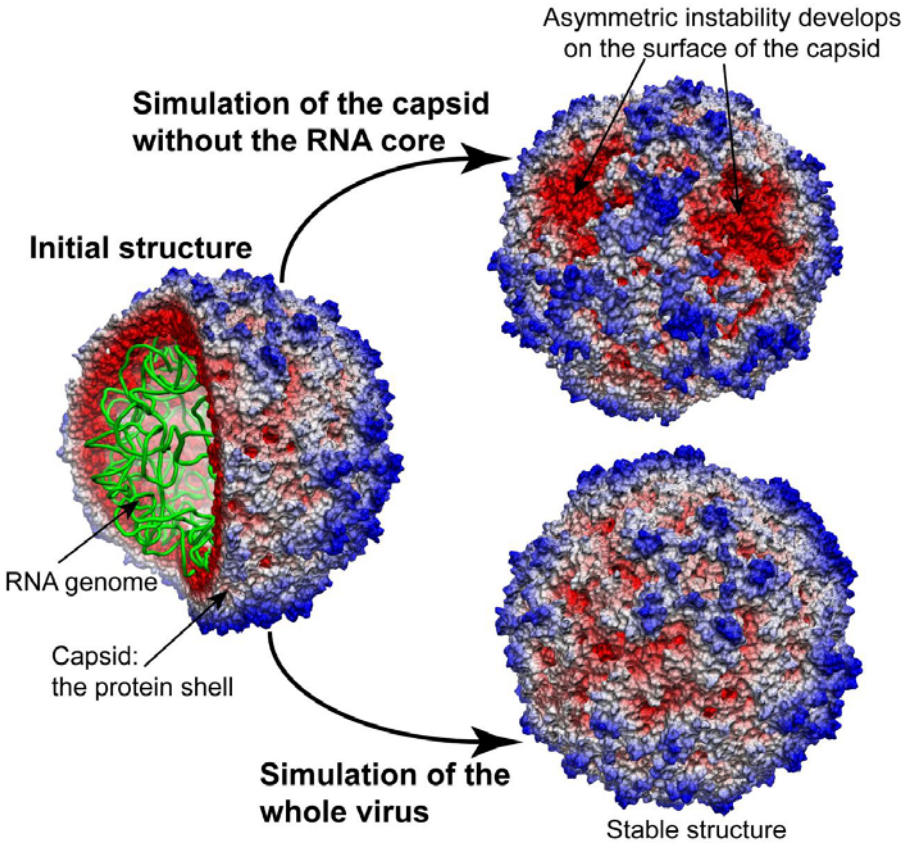
Shows conversion of PrP protein from healthy to harmful isoform. Aggregation of misfolded *intermediates* appears to be the pathogenic species in amyloid (e.g. "mad cow" & Alzheimer's) diseases.

Note: this could *only* have been discovered with simulation!

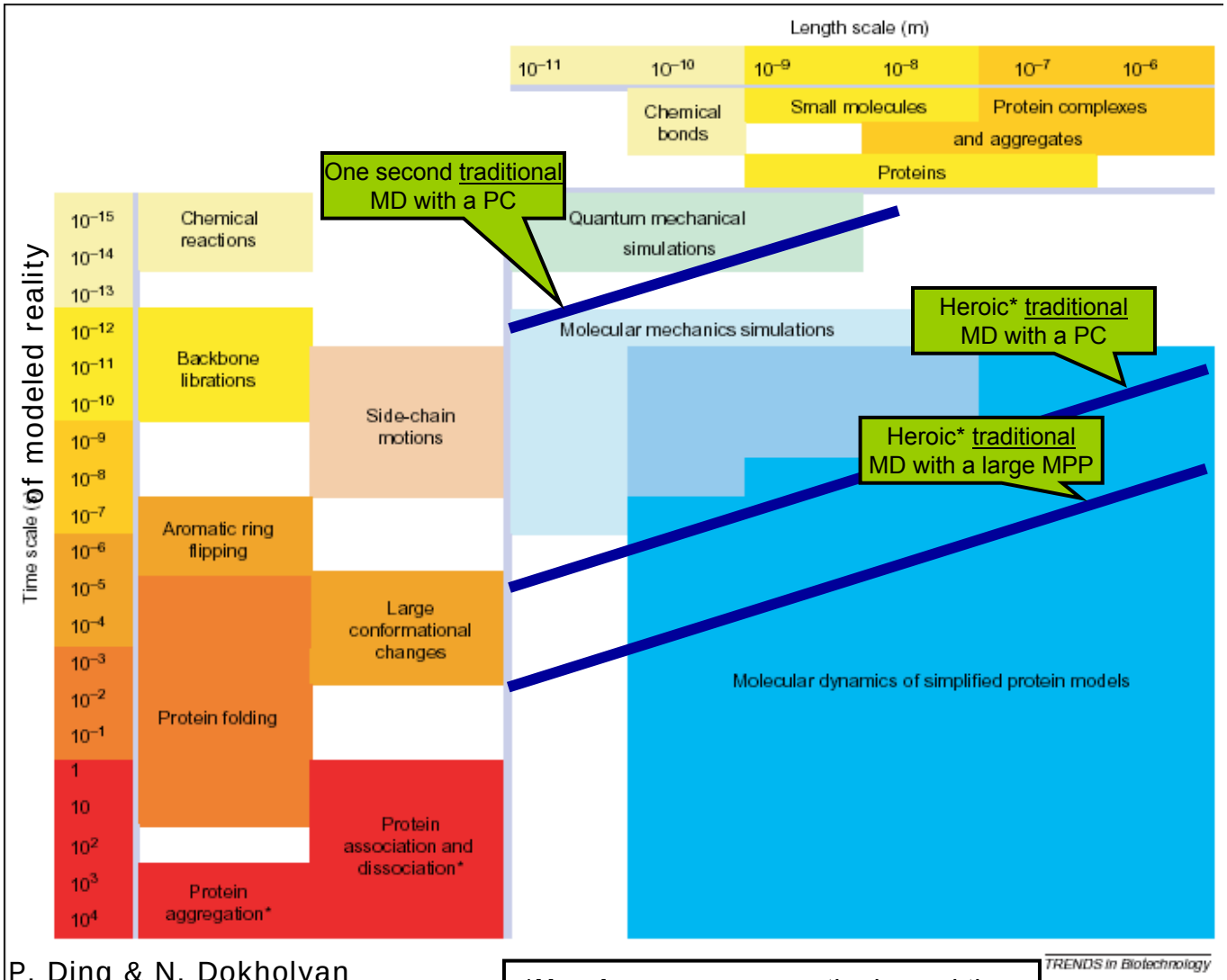
Why *LARGE* MD Simulations are so important …

MD simulations are often "heroic":

100 days on 500 nodes …



Motivation - Why *Accelerate* MD?



P. Ding & N. Dokholyan
Trends in Biotechnology,2005

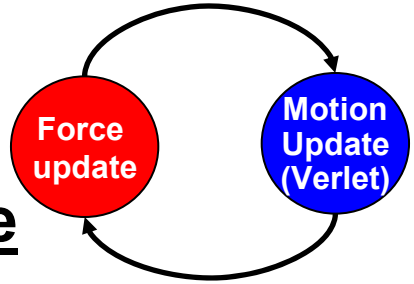
*__Heroic__ ≡ > one month elapsed time

What is (Traditional) Molecular Dynamics?

MD – An iterative application of Newtonian mechanics to ensembles of atoms and molecules

Runs in phases ➔

state of each particle is updated every fs



Generally O(n), done on host

Initially O(n²), done on coprocessor

Many forces typically computed, $F^{total} = F^{bond} + F^{angle} + F^{torsion} + F^{H} + F^{non-bonded}$

but complexity lies in the non-bonded, spatially extended forces:
van der Waals (LJ) and Coulombic (C)

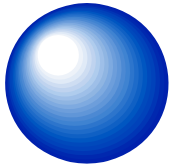
$$F_i^{LJ} = \sum_{j \neq i} \frac{\varepsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \vec{r}_{ji}$$

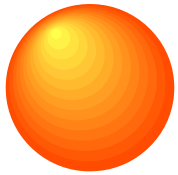
$$F_i^{C} = q_i \sum_{j \neq i} \left(\frac{q_i}{|r_{ji}|^3} \right) \vec{r}_{ji}$$

An Alternative ...

Only update particle state when "something happens"

- **"Something happens" = a discrete event**

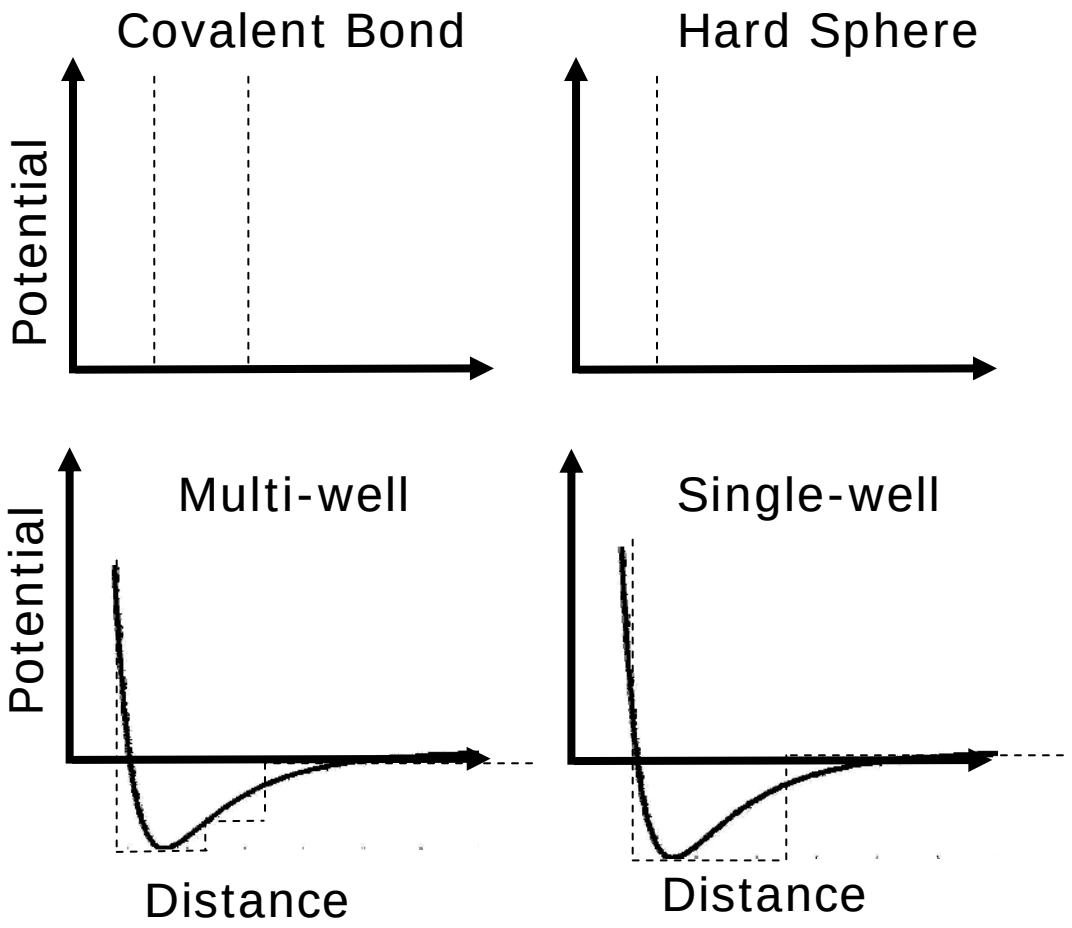




- **<u>Advantage</u> → DMD runs 10^6 times faster than tradition MD**
- **<u>Disadvantage</u> → Laws of physics are continuous**

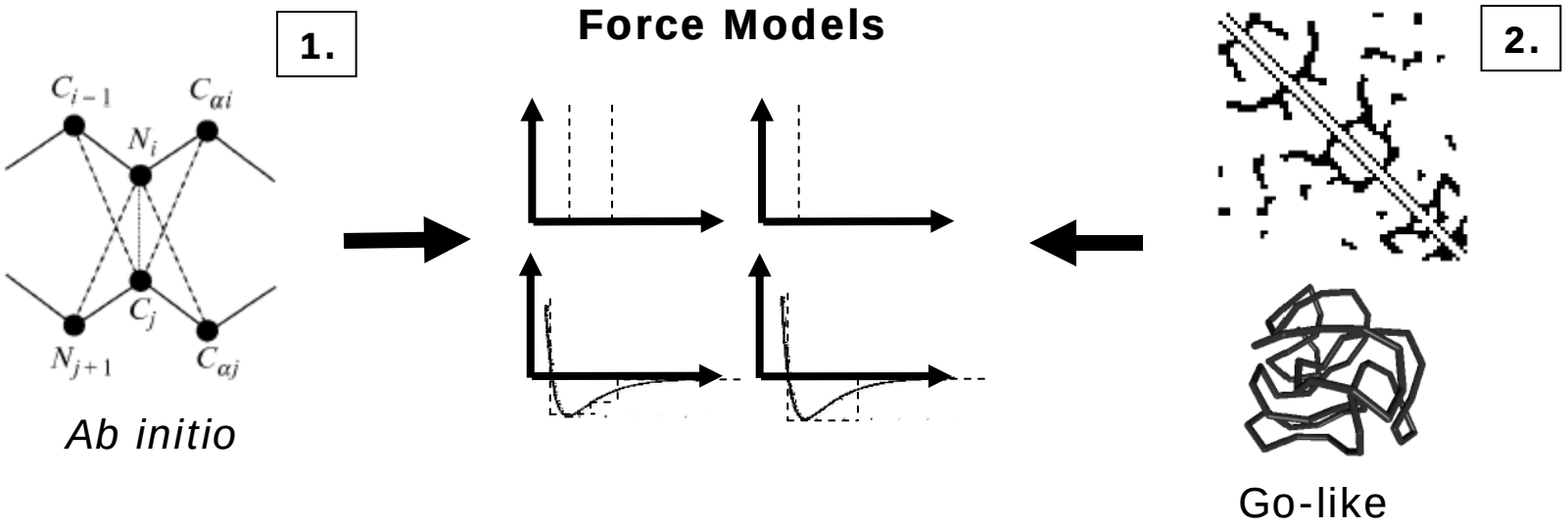
But the physical world isn't discrete …

DMD force approximation



While we're approximating forces …

- Traditional MD often uses all-atom models
- DMD often models atoms behaviorally
  1. *Ab initio,* assuming no knowledge of specific protein dynamics
  2. *Go*-like models, which use empirical knowledge of the native state



1. Urbanc et al. 2006
2. Dokholyan et al. 1998

After all this approximation …

… is there any reality left??

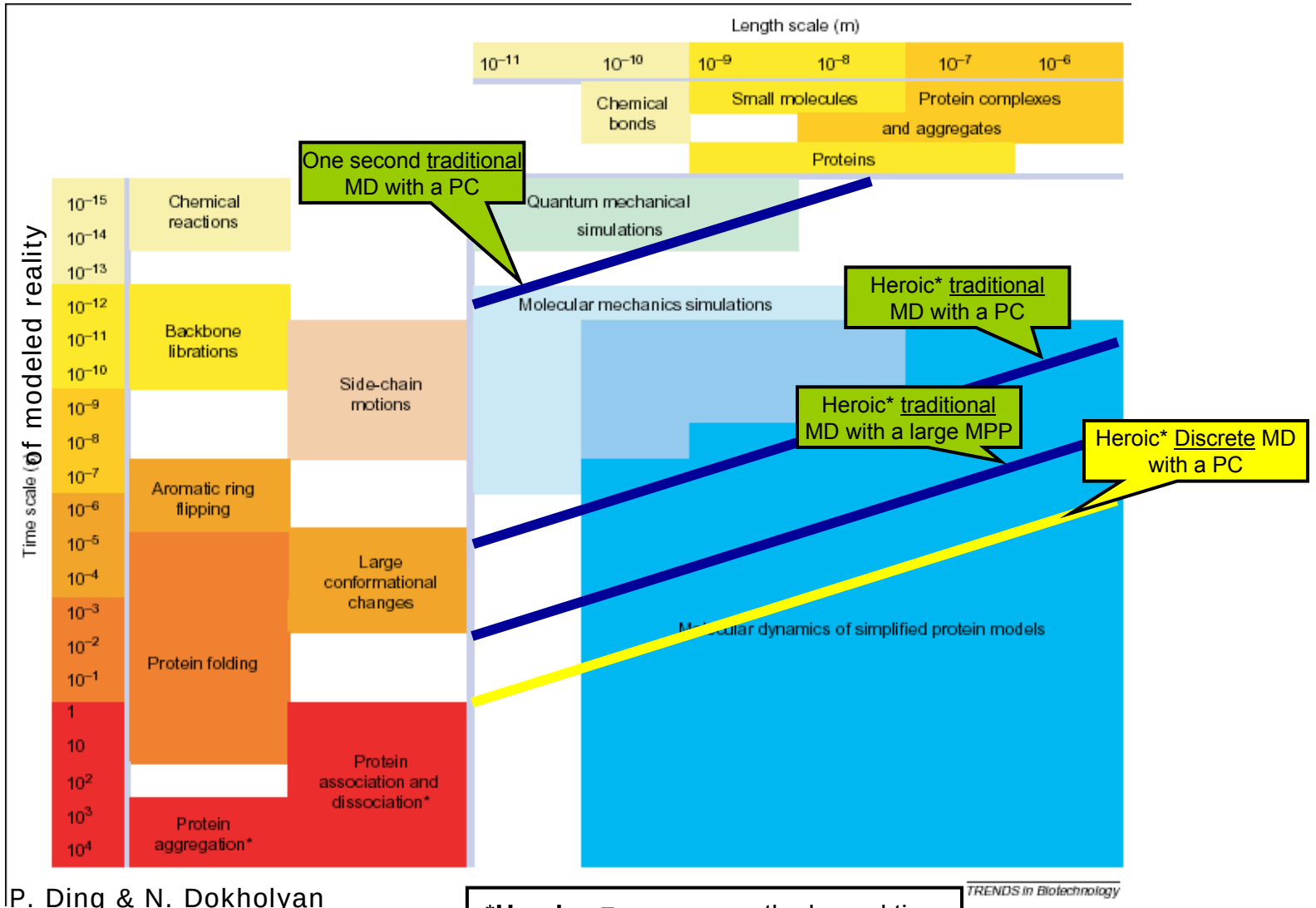
Yes, but

requires *application-specific model tuning*

- **Using traditional MD**
- **Frequent user feedback**

→ ***Interactive simulation***

Current DMD Performance



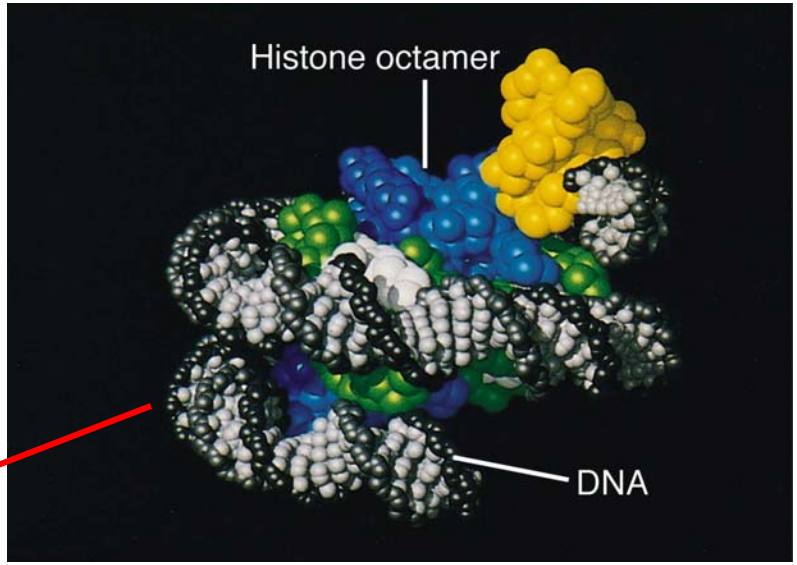
P. Ding & N. Dokholyan
Trends in Biotechnology,2005

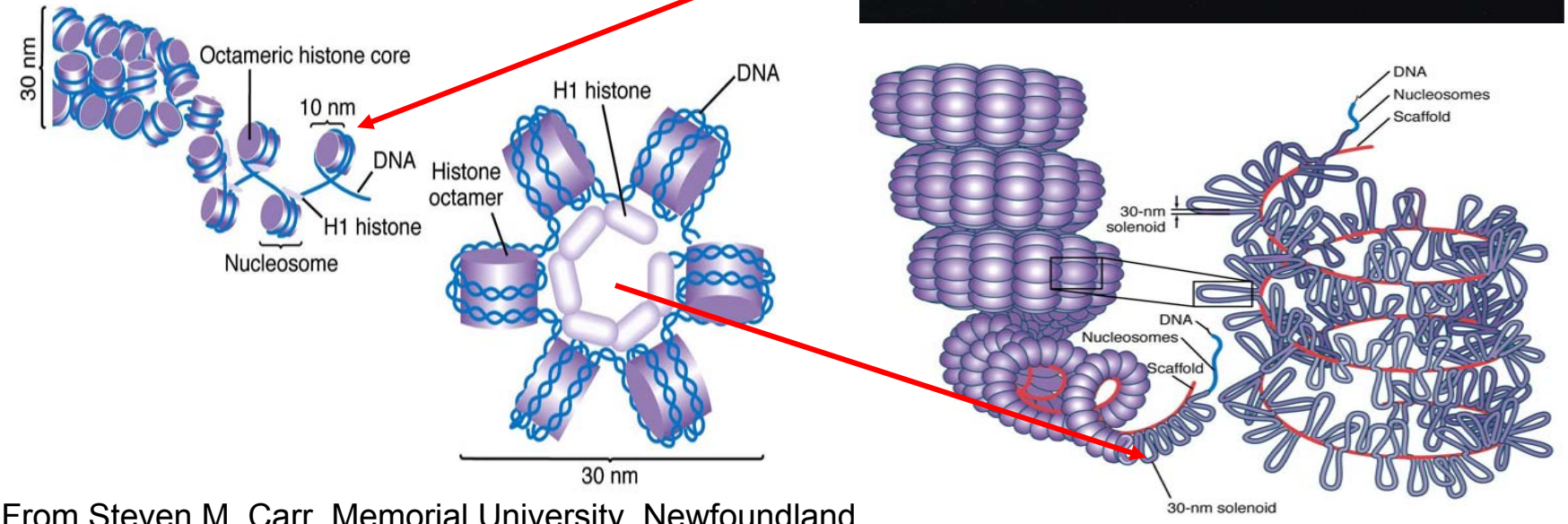
Heroic ≡ > one month elapsed time

Motivation - Why *Accelerate* DMD?

Example: Model nucleosome dynamics

i.e., *how DNA is packaged and accessed – three meters of it in every cell!*

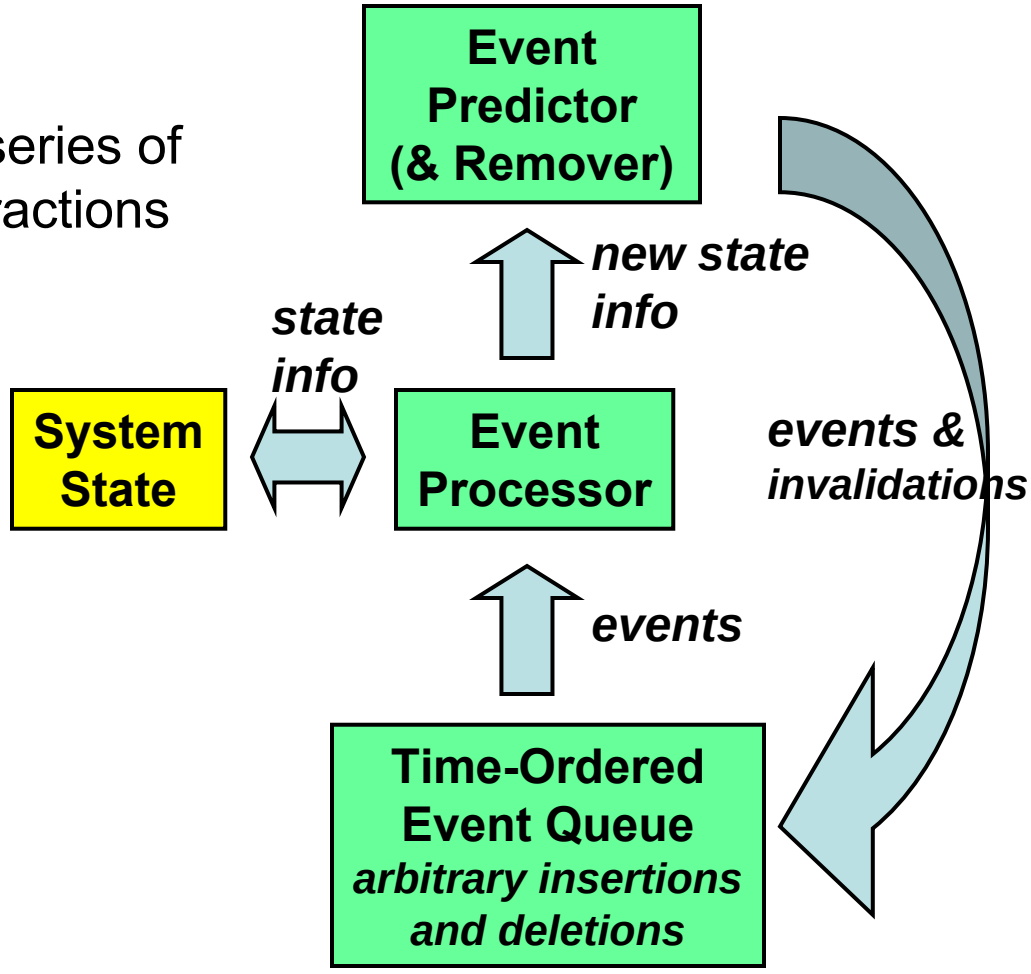




From Steven M. Carr, Memorial University, Newfoundland

Discrete Event Simulation

- Simulation proceeds as a series of discrete element-wise interactions
  - **NOT** time-step driven



- Seen in simulations of …
  - Circuits
  - Networks
  - Traffic
  - Systems Biology
  - Combat

How to make DMD even faster? Parallelize??

Approaches to Parallel DES are well known:

- **Conservative**
  - Guarantees causal order between processors
  - Depends on “safe window” to avoid serialization
- **Optimistic**
  - Allows processors to run (more) independently
  - Correct resulting causality violations with rollback

Neither approach has worked in DMD:

- Conservative: no safe window → ***causal order = serialization***
- Optimistic: → ***rollback is frequent and costly***

No existing production PDMD system!

What's hard about parallelizing DMD?

DMD production systems are highly optimized

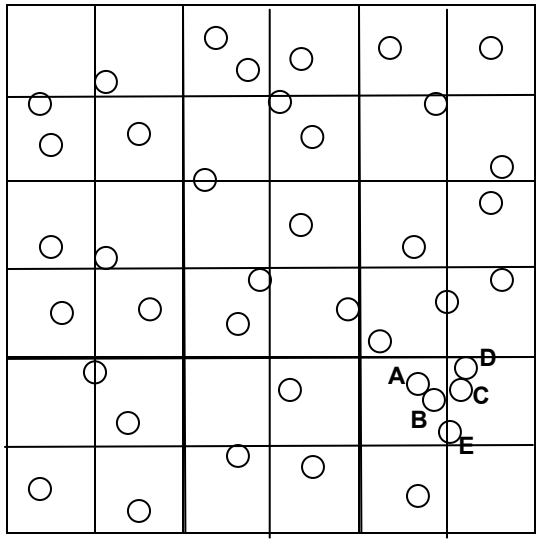
- 100K events/sec for up to millions of particles (10us/event)
- Typical message passing latency ~1us-10us
- Typical memory access latency ~ 50ns-100ns

What's hard about parallelizing DMD?

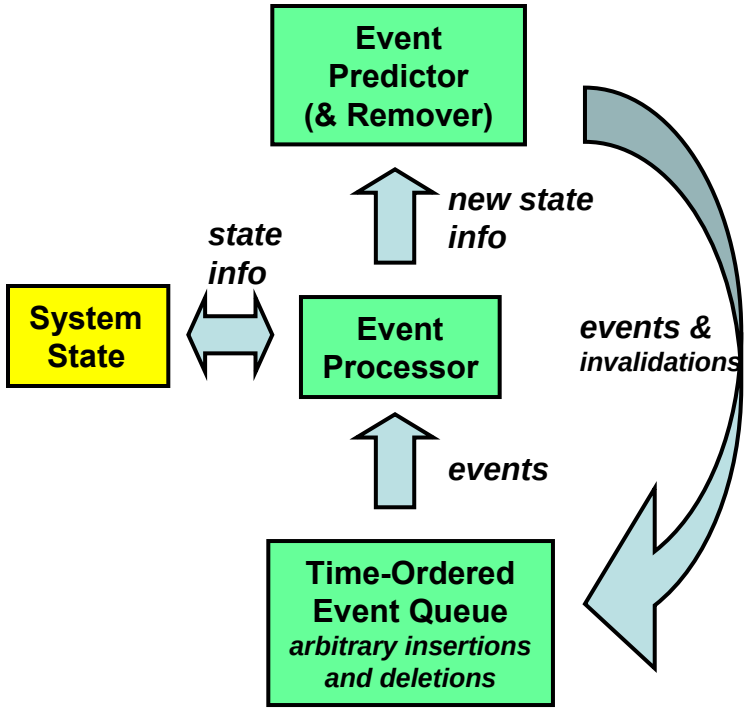
How about Task-Based Decomposition?

New events can

- invalidate queued events anywhere in the event queue
- be inserted anywhere in the event queue



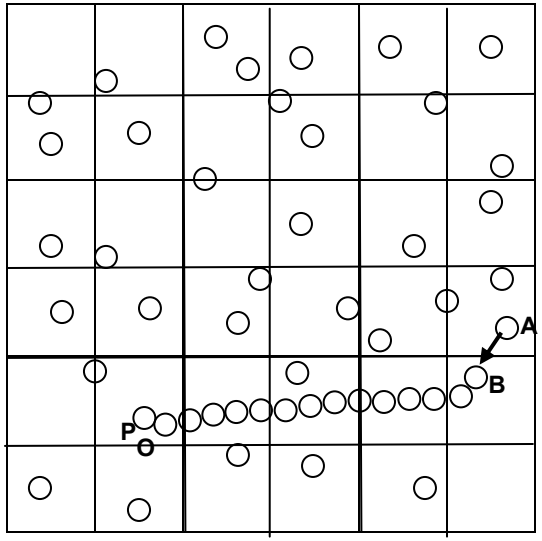
After events **AB** and **CD** at t_0 and $t_{0+\varepsilon}$, newly predicted event **BC** happens almost immediately – inserted at head of queue! Also, previously predicted **BE** gets cancelled.



What's hard about parallelizing DMD?

But those events were necessarily local --

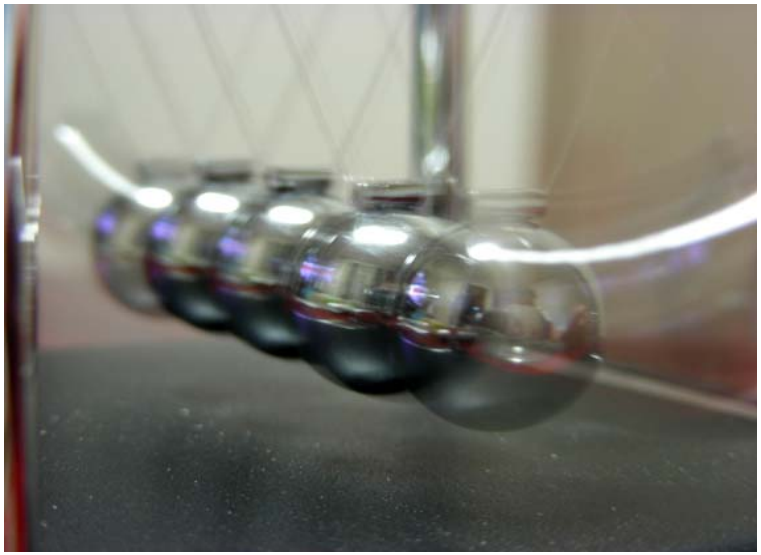
<u>Can't we partition the simulated space?</u>



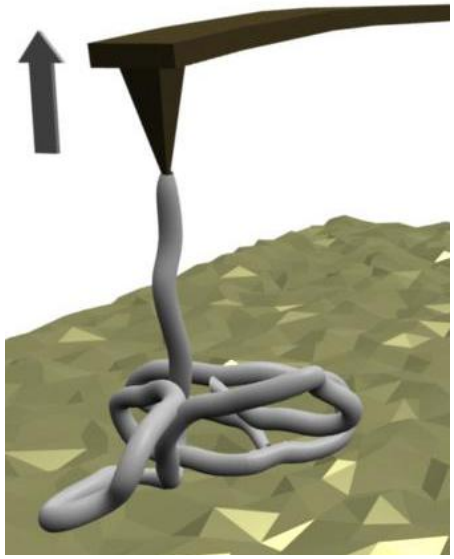
After event **AB,** cascade of events causes **OP** to happen almost immediately on the other side of the simulation space.

Yes, but requires speculation and rollback

Event propagation can be *infinitely fast* over any distance!



Note: "chain" with rigid links is analogous and much more likely to occur in practice



Atomic Force Microscope unravels a protein

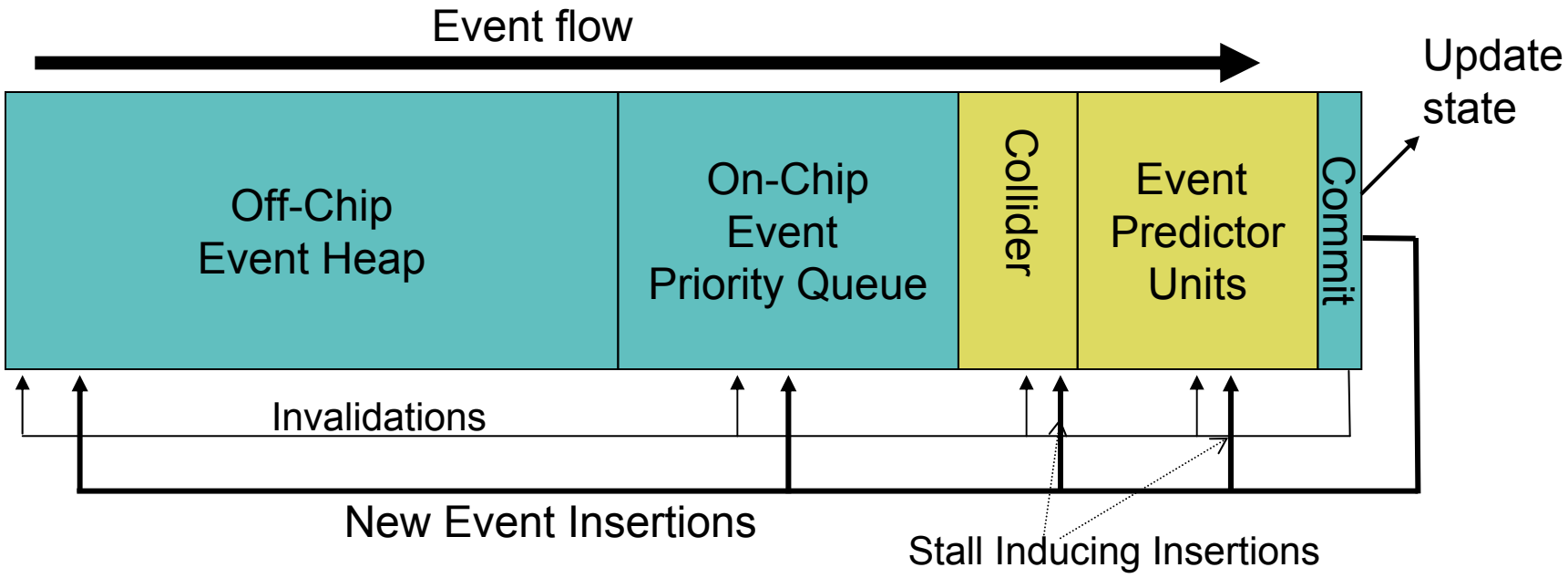
Outline

- Overview: MD, DMD, DES, PDES
- **FPGA Accelerator conceptual design**
  - **Design overview**
  - **Component descriptions**
- Design Complications
- FPGA Implementation and Performance
- Multicore DMD
- Discussion

FPGA Overview - Dataflow

Main idea: DMD in one big pipeline

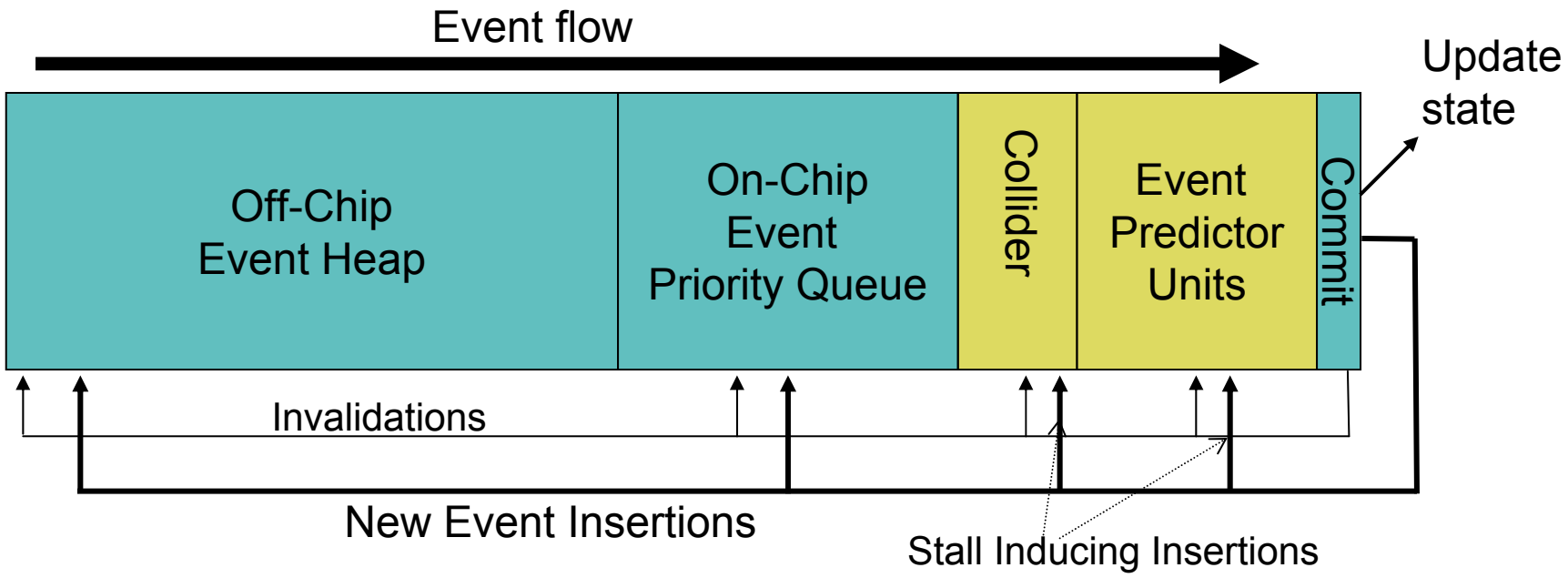
- Events processed with a throughput of one event per cycle
- Therefore, ***in a single cycle***:
  - State is updated (event is **committed**)
  - Invalidations are processed
  - New events are inserted – up to four are possible



FPGA Overview - Dataflow

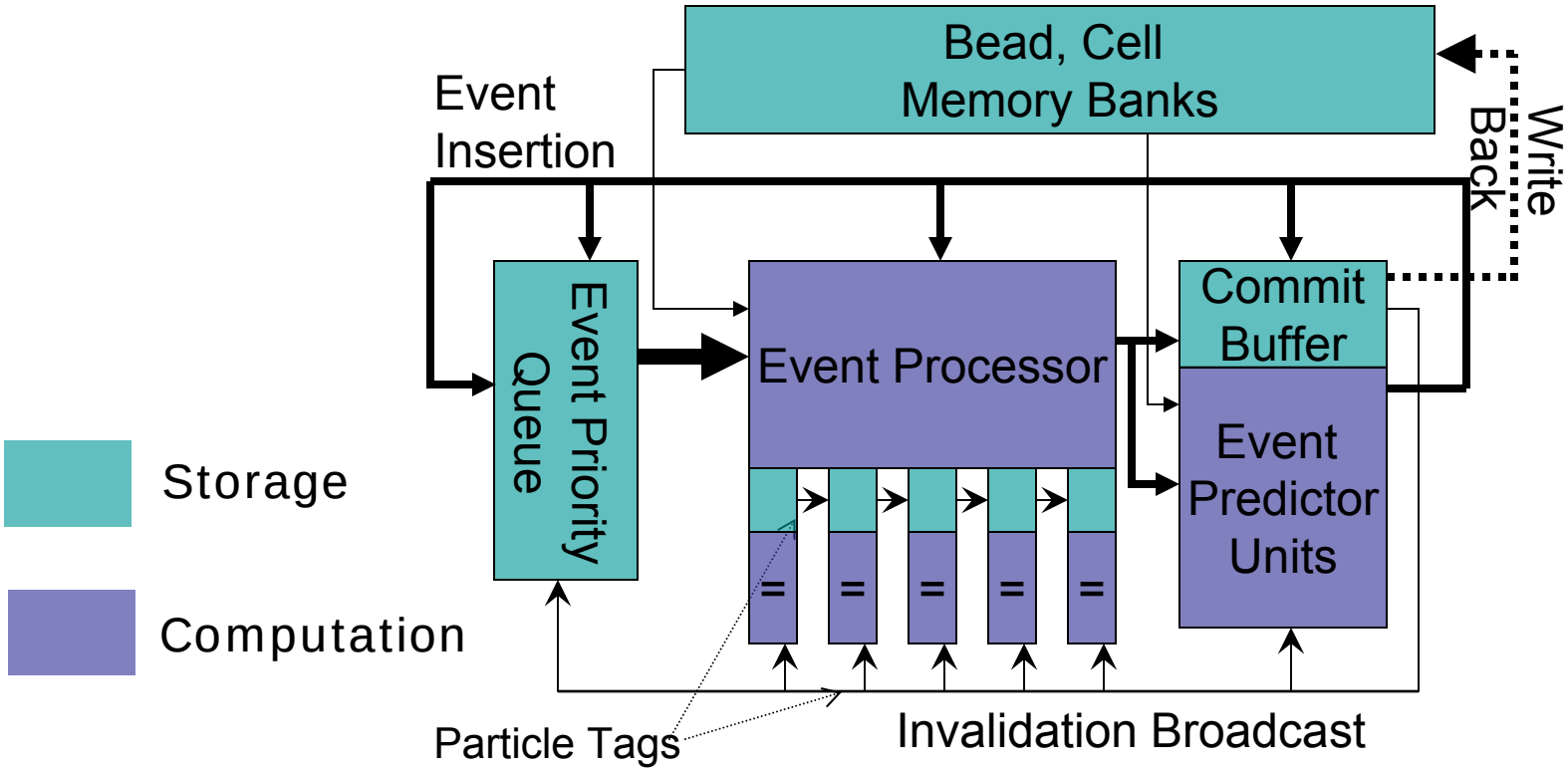
Main idea: DMD in one big pipeline

- Events processed with a throughput of one event per cycle
- ***Three complications:***
  1. Processing units must have flexibility of event queue
  2. Events cannot be processed using stale state information
  3. Off-chip event queue must have same capability as on-chip

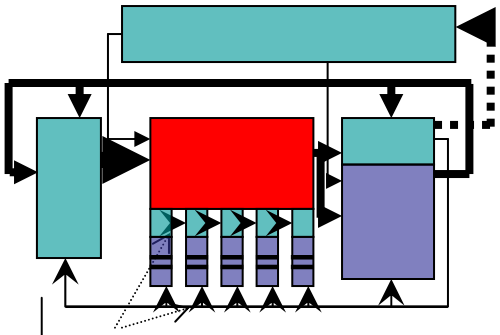


Components

High-Level DMD Accelerator System Diagram

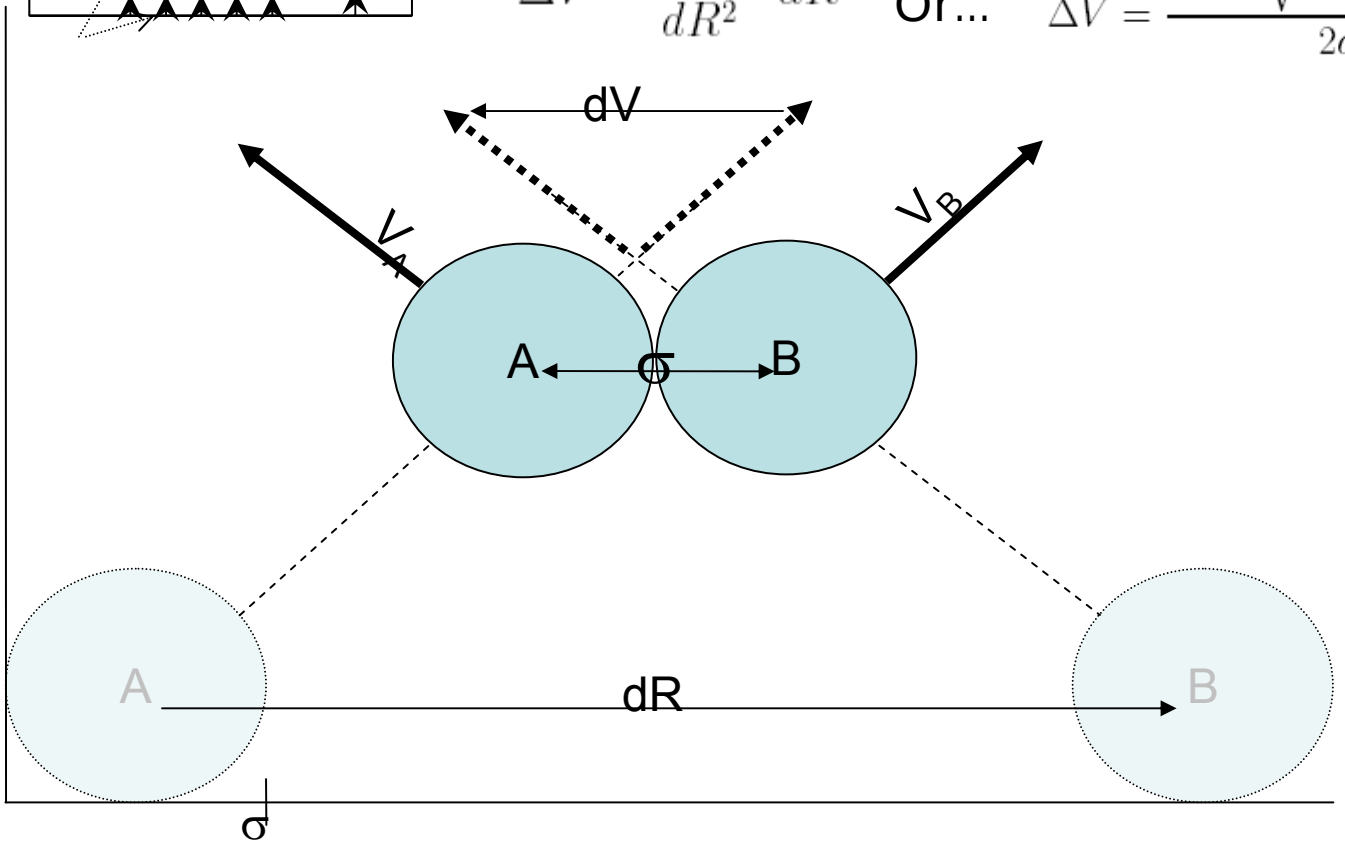


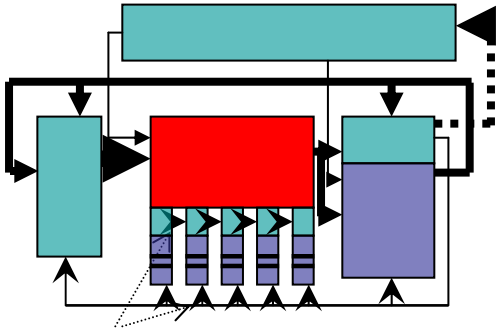
Event Processor



Fetch two beads' motion parameters and process to compute new motion parameters

$$\Delta V = \frac{\pm b}{dR^2} \cdot dR \quad \text{Or...} \quad \Delta V = \frac{-b \pm \sqrt{b^2 - 4dR^2(\frac{\Delta\mu}{m})}}{2dR^2} \cdot dR$$





Event Processor – Notes

- Straightforward computational pipelines
- Several event types are possible
  - Hard sphere collisions
    - Billiard balls, atoms at vdW radius
  - Hard bond collisions
    - Links on chain, covalent bonds
  - Soft interactions
    - v.d.W. forces

Hydrogen bonds will provide a new challenge …

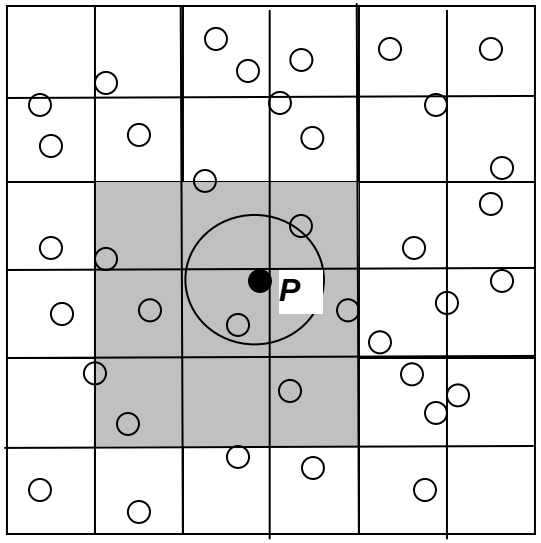
Make Prediction *O(N)* with Cell Lists

$$F_i^{LJ} = \sum_{j \neq i} \frac{\varepsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \vec{r}_{ji}$$

Observation:

- Typical volume to be simulated = 100Å^3
- Typical LJ cut-off radius = 10Å

Therefore, for all-to-all O(N^2) computation, *most work is wasted*



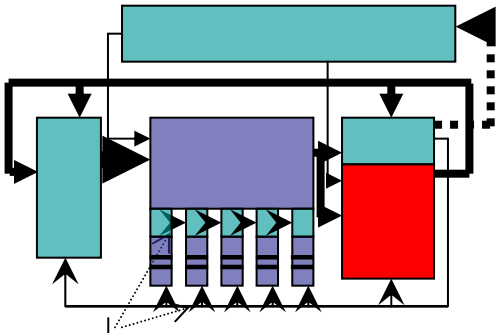
Solution:

Partition space into "cells," each roughly the size of the cut-off

Predict events with ***P*** *only w.r.t. beads in adjacent cells.*

- Issue → shape of cell – spherical would be more efficient, but cubic is easier to control
- Issue → size of cell – smaller cells mean less useless force computations, but more difficult control. Limit is where the cell is the atom itself.
- **For DMD, cell size ~ bead size**

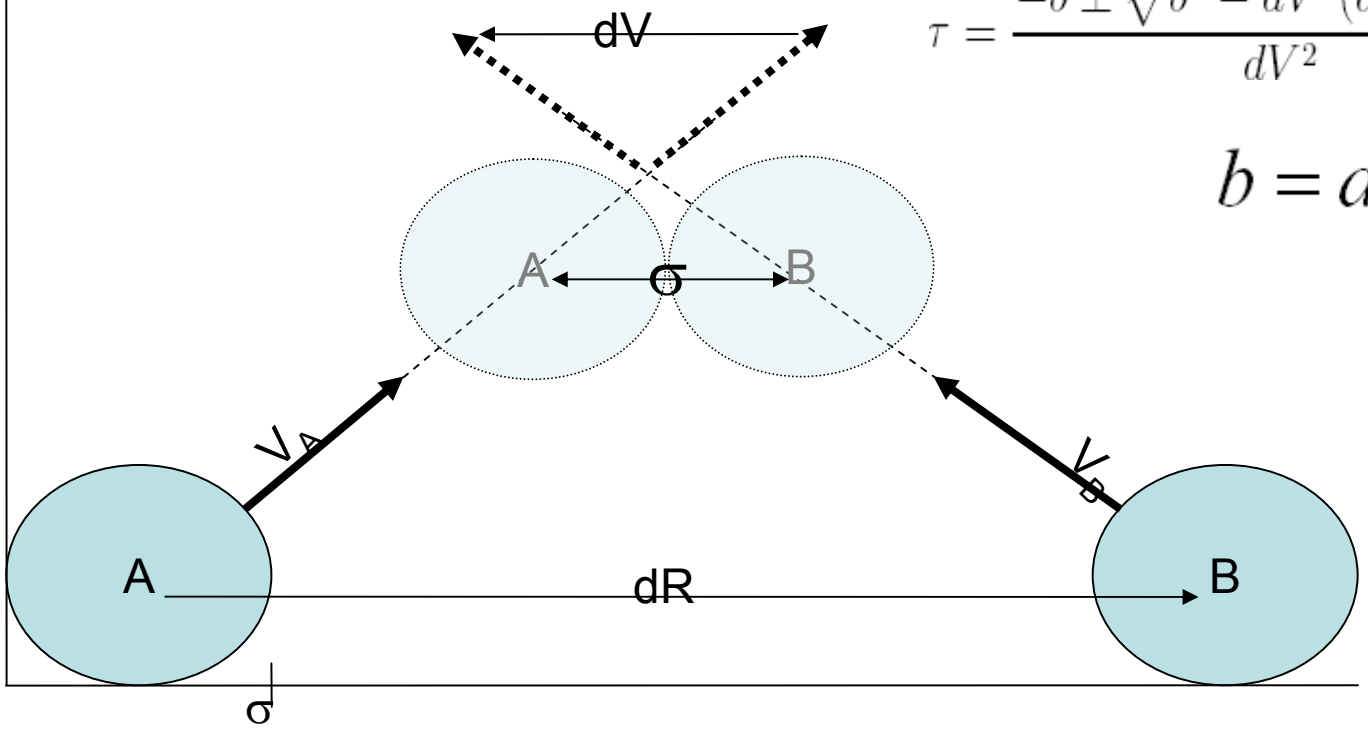
Event Predictor



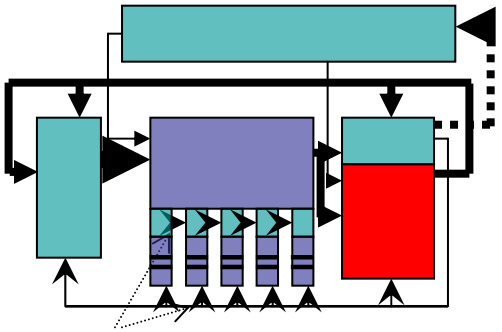
For each bead just processed:
For each bead in the neighboring cells
Fetch motion parameters and process to compute time/type of (possible) new event

$$\tau = \frac{-b \pm \sqrt{b^2 - dV^2(dR^2 - \sigma^2)}}{dV^2}$$

$$b = dV \bullet dR$$



Work for Event Predictor

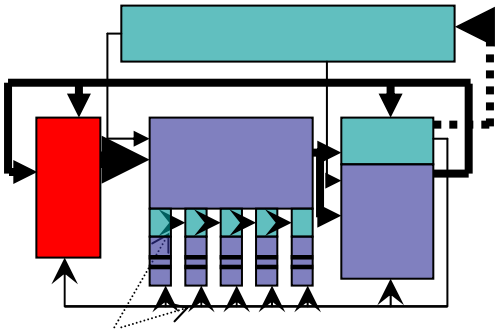


For each bead just processed:
For each bead in the neighboring cells
Fetch motion parameters and process to compute time/type of (possible) new event

Beads per collision-type event → 2
Cells per neighborhood → 27 – 46
Beads per cell → 0 – 8
Beads per neighborhood → 0 – 100
Typical # of beads/neighborhood → 5

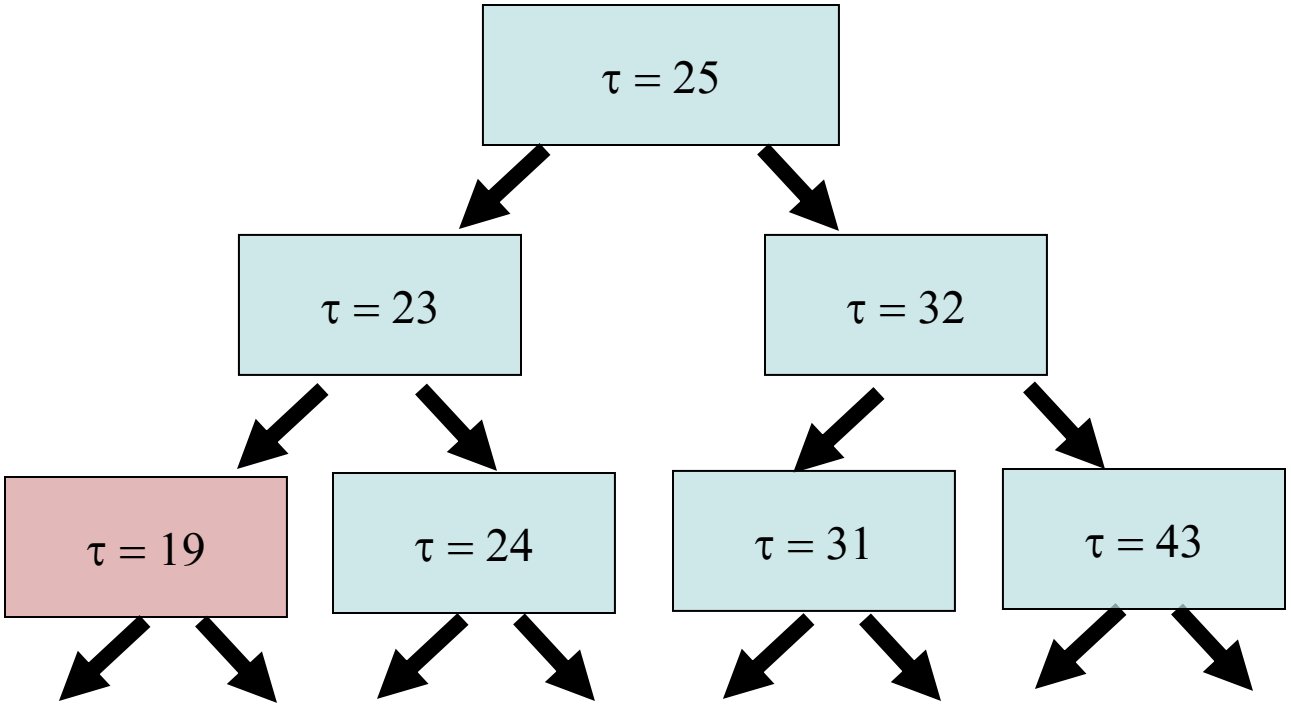
Number of predictor units to maintain throughput
→ 10+ required, 16 desired

Event Calendar (queue)

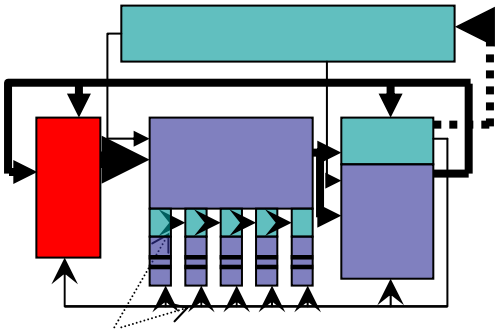


In serial implementations, data structures store future events. Basic operations:

1. Dequeue next event
2. Insert new events
3. Delete invalid events

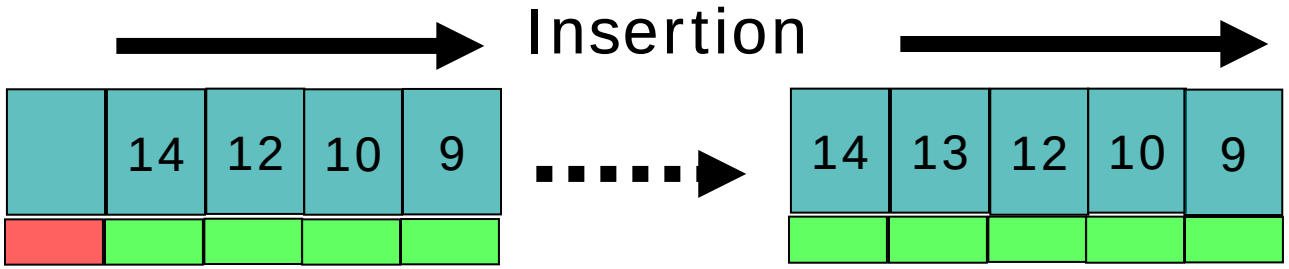


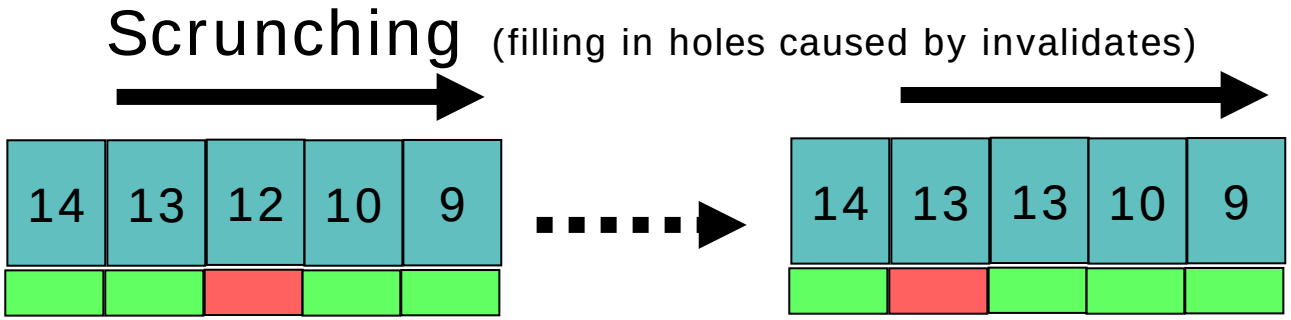
Event Calendar Priority Queue



Basic capabilities for every cycle:

1. Advance events one slot if possible
2. Insert a new event into an arbitrary slot as indicated by time tag
3. Record arbitrary number of invalidations as indicated by bead tag
4. Fill in holes caused by invalidations (scrunching) by advancing events extra slot when possible





Priority Queue Performance: Intuition

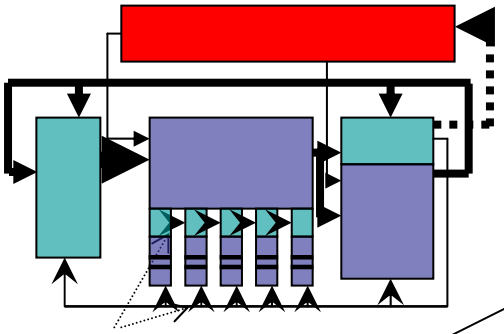
Question: With events constantly being invalidated, what is the probability that a "hole" will reach the end of the queue, resulting in a payloadless cycle?

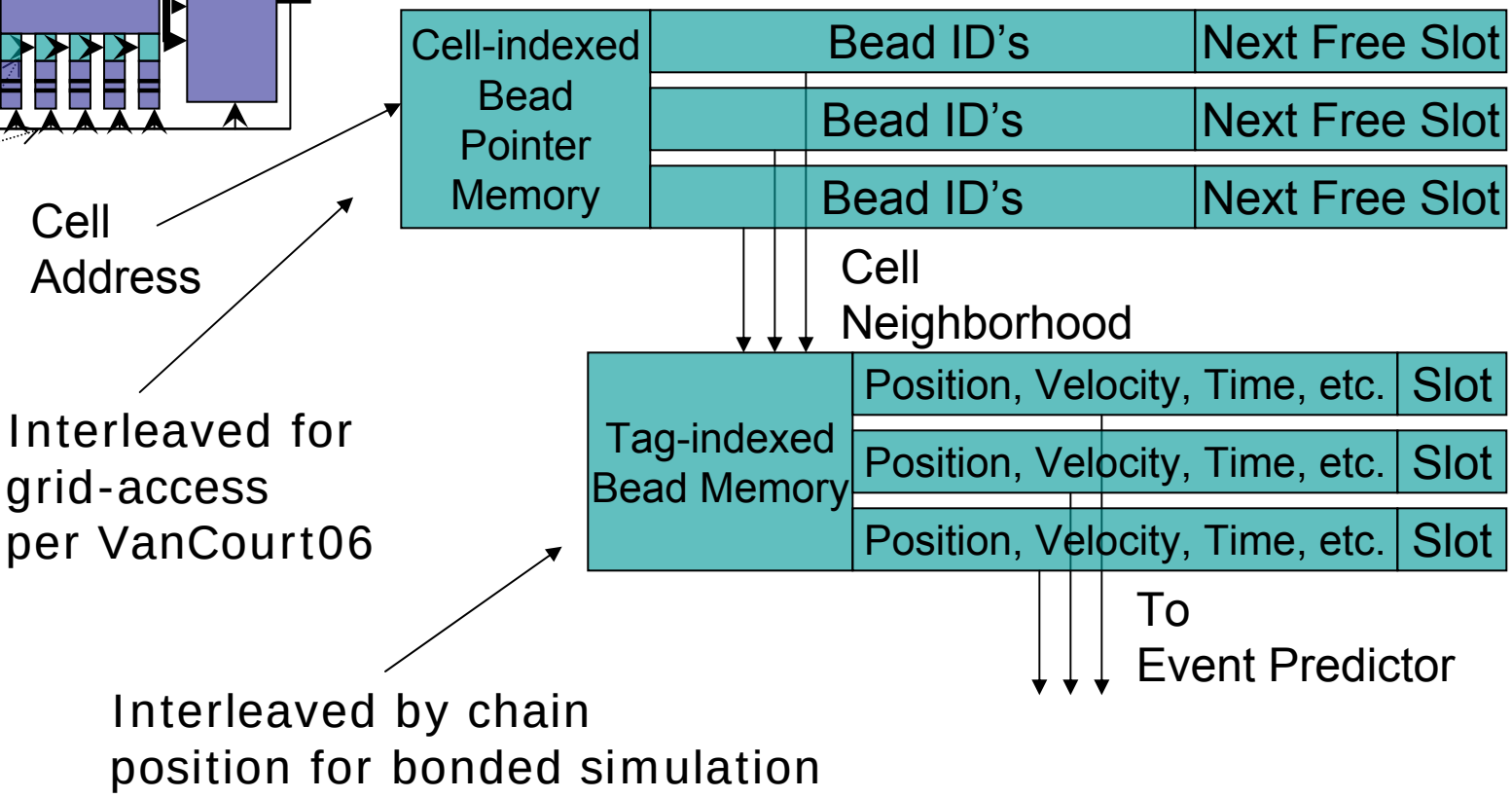
Observations:

1. There is a steady state between insertions and invalidations/commitments
2. Scrunching "smoothes" disconnect between insertions and invalidations
3. Insertions and invalidations are uniformly distributed
4. Scrunching not possible for compute stages

Empirical result: < .1% of cycles (non-stalls) commit holes

Bead/Cell Memory Organization – a.k.a., State





Back to event prediction

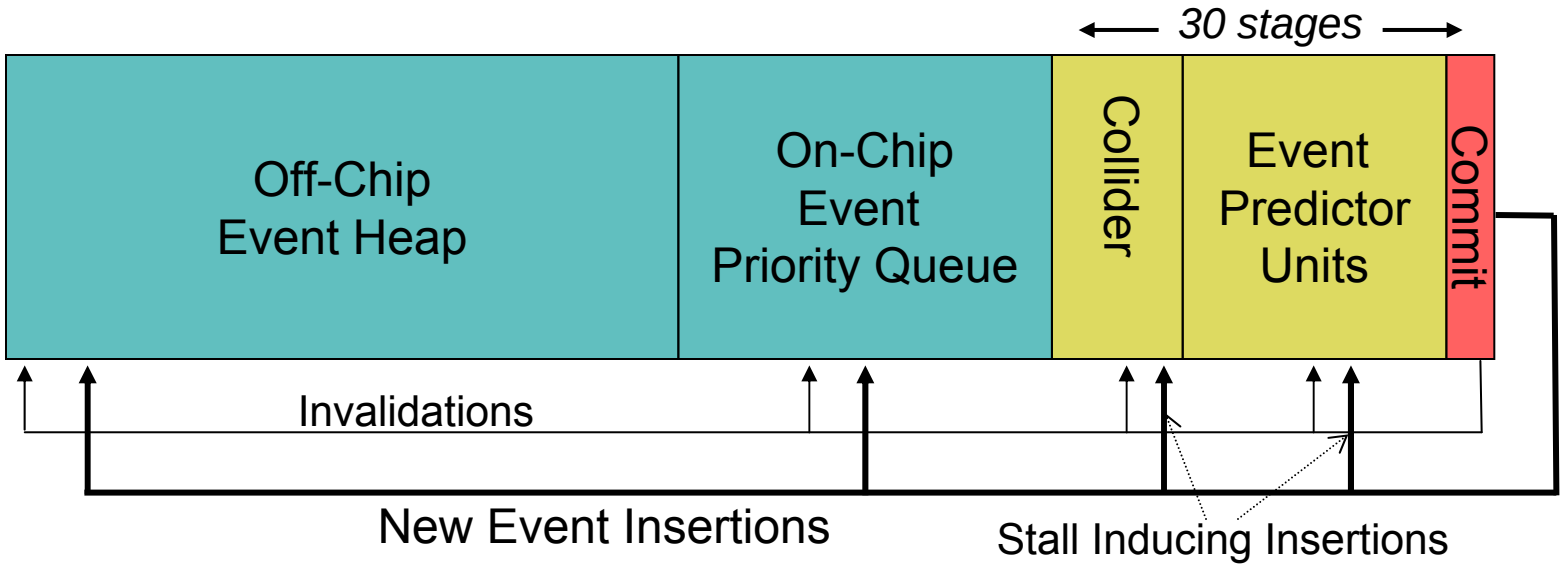
- Organize Bead and Cell list memory so that prediction can be fully pipelined
  - Start with bead in cell x,y,z
  - For each neighboring cell, fetch bead IDs
  - For each bead ID, fetch motion parameters
  - Schedule these beads with x,y,z to event predictors
  - Of events predicted, sort to keep only soonest

Outline

- Overview: MD, DMD, DES, PDES
- FGPA Accelerator Conceptual Design
- **Design Complications – Dealing with …**
  - **Causality Hazards**
  - **Coherence Hazards**
  - **Large Models with finite FPGAs**
- FPGA Implementation and Performance
- Multicore DMD
- Discussion

Causality Hazards

Observation: New events can need to be inserted *anywhere* in the pipeline
Observation: This includes "processing stages" of the pipeline
Problem: if an event is inserted into a processing stage, it will have skipped some of its required computation (event processing or event prediction)
Solution, part 1: all events must be inserted into the first processing stage, even if that is many stages earlier than where it belongs
Another Problem: now the events are out of order
Solution, part 2: stall pipeline until newly inserted event "catches up"
For processing stages, this requires a set of shadow registers



Causality Hazards – Performance Hit

- Insertions are uniformly distributed in the event queue
- Queue size > 10,000 events

➔ P(hazard per insertion) < 30/10,000 = .3%

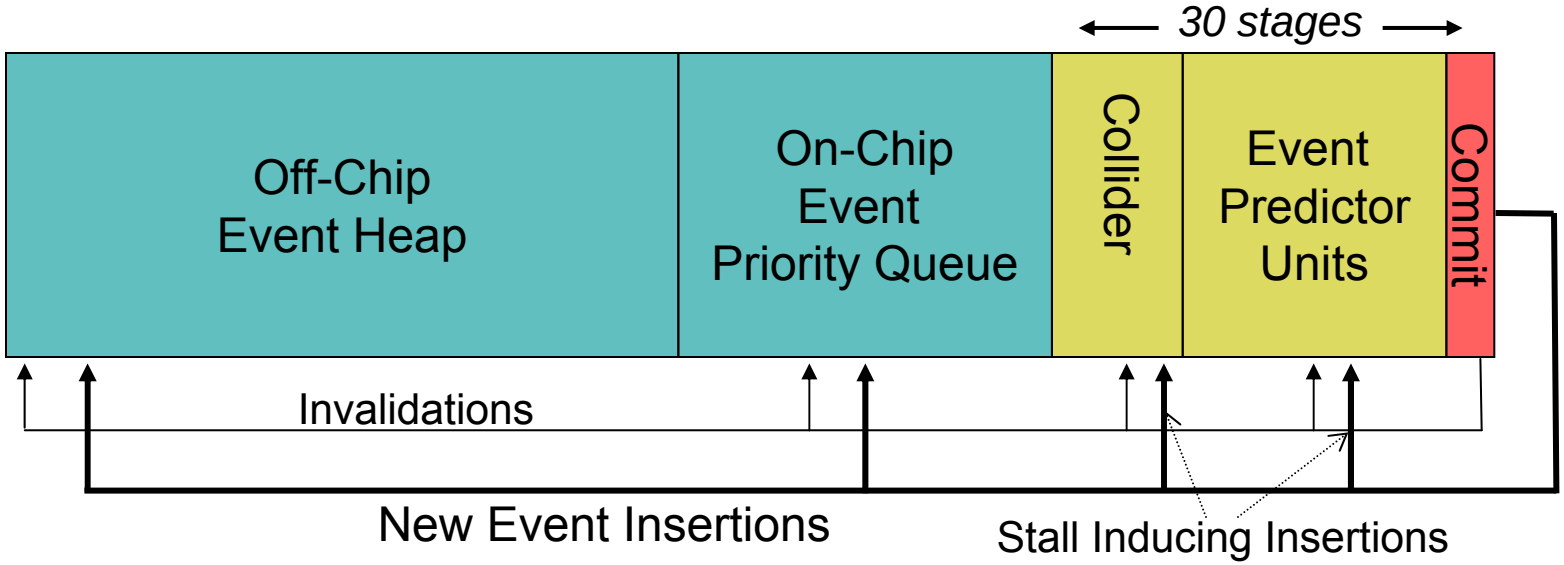
- 2.3 insertions (new events) per commitment

➔ P(hazard per commitment) < .7%

- Stall cycles per hazard ~ 15

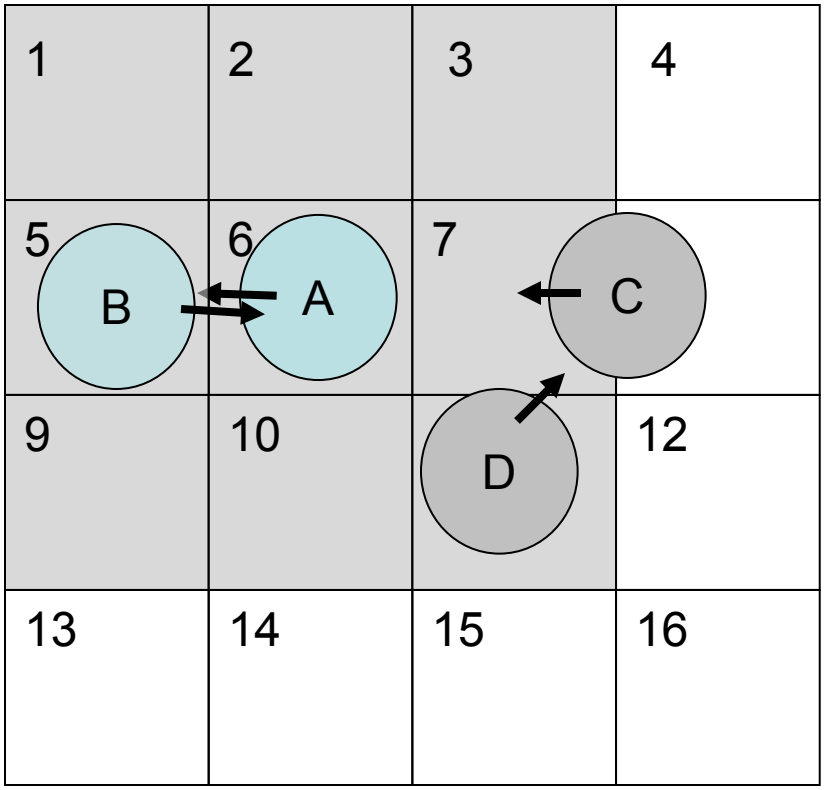
➔ Expected Stalls per Commitment < .011

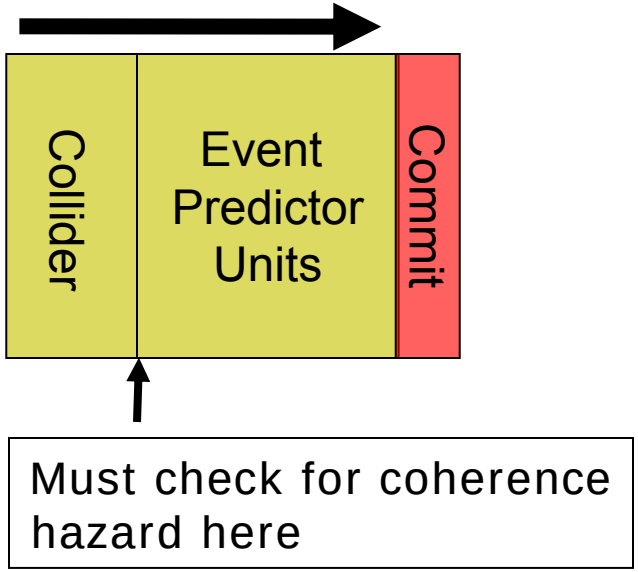
➔ Performance loss due to causality stalls ~ 1%



Coherence Hazards

- Bead A finishes in collider (event **AB**) and looks at particles in its neighborhood for possible new events.
- If processing continues, it sees it will collide with particle C (event **AC**)
- But particle C has ***already*** collided with particle D (event **CD**)
- **PROBLEM: A is predicting AC with stale data (AD should be predicted).**

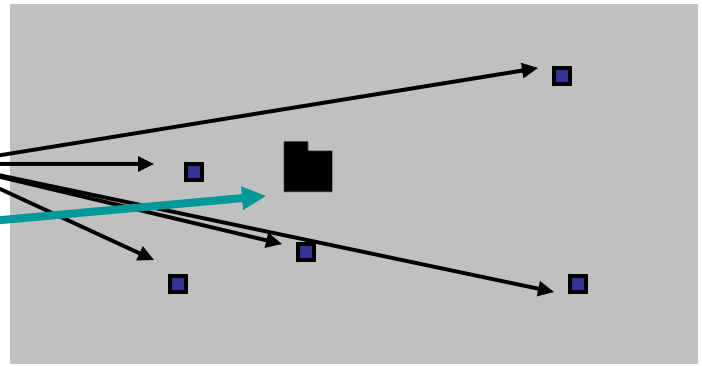




Dealing with Coherence Hazards

Maintain bit vector of cells in the simulation space with events in the predictor

Example →
- locations of events in predictor
- location of region of new event entering predictor



For each bead entering predictor:

➔ *Is there a bead ahead of me in my neighborhood?*

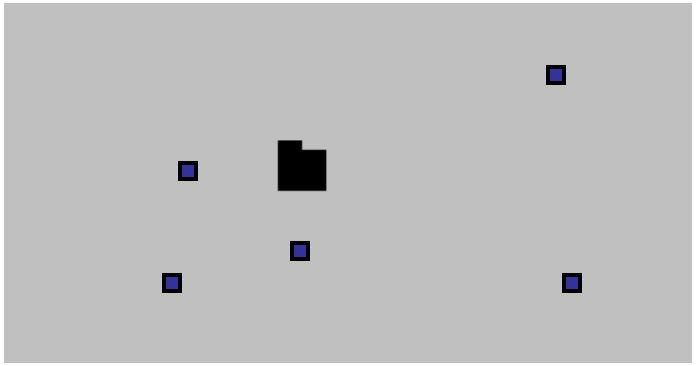
IF TRUE, THEN ***Coherence Hazard!***

STALL until event is committed

Coherence Hazards – Performance Hit

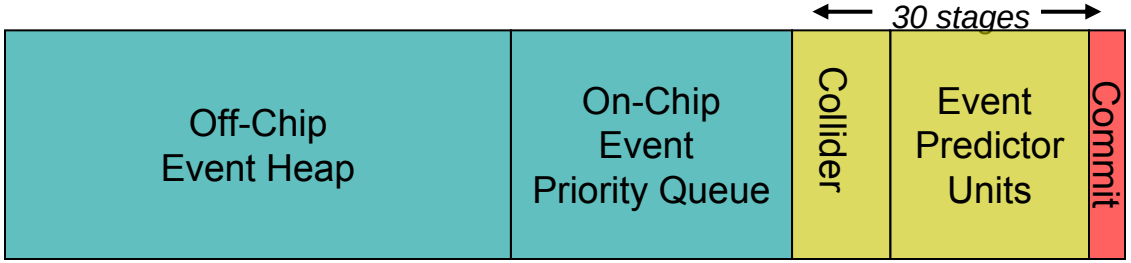
- Events are uniformly distributed in space
- Neighborhood size = 27 cells
- 23 stages in predictor
- Simulation space is typically 32x32x32
- Cost of a coherence hazard = 23 stalls
- Probability of a coherence hazard →

  27 Cells * 23 Stages / 32x32x32 Cells = 1.8%
- Performance hit of coherence hazard ~ 40%



Complication of a complication

What about causality hazards that are also coherence hazards?



Scenario →

- New event E needs to be inserted into a "computation" slot
- Events in the computation slots are set aside while E catches up.
- Potential problem: what if there is an element with a time tag later than E that got set aside while E caught up, but which causes a coherence hazard with E?

Solution → restart computations of all events in computation slots on causality hazards. Clear scoreboard.

Off-chip Event Calendar

- Recall: must be able to queue, dequeue, and invalidate events – all with a throughput of 100Mhz

- Problem: off-chip memory is not amenable to design just presented
  - no broadcast, independent insertion, …
  - Performance is O(log N)

- What we have going for us:
  - Don't need the events any time soon *>> Trade off time for bandwidth?*
  - FPGAs are slow
  - FPGAs have massive off-chip bandwidth *>> only a fraction of the on-chip*
  - Easy to implement separate controllers for several off-chip memory banks

Serial Version – O(1) Priority Queue

Observation (from serial version – G. Paul 2007)

- A typical event calendar has thousands of events, but only a few are going to be used soon
- This makes the N in O(log N) performance much larger than it needs to be

Idea:

- Only use tree-structured priority queue for events that are about to happen
- Keep other events in unsorted lists, each representing a fixed time interval some time in the future

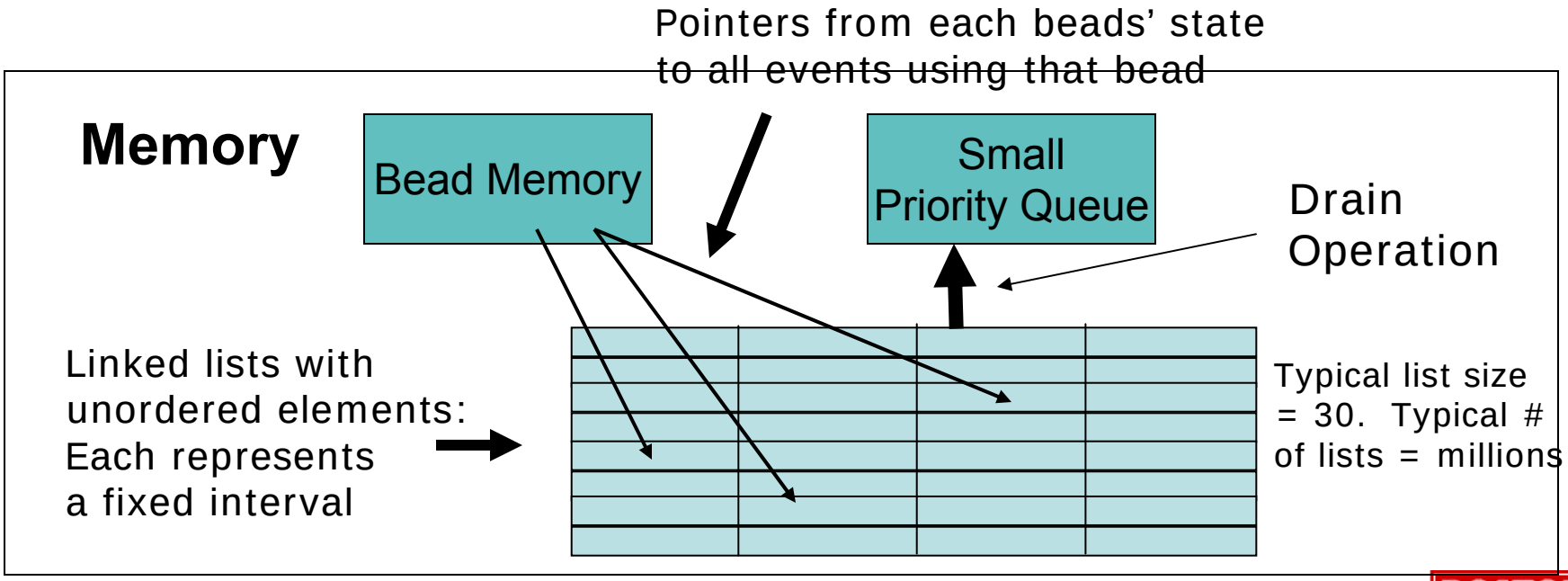
Serial Version – Operation

Dequeue next – take from head of priority queue

Insert events – if not very soon, then time tag determines the list to which the event is to be appended

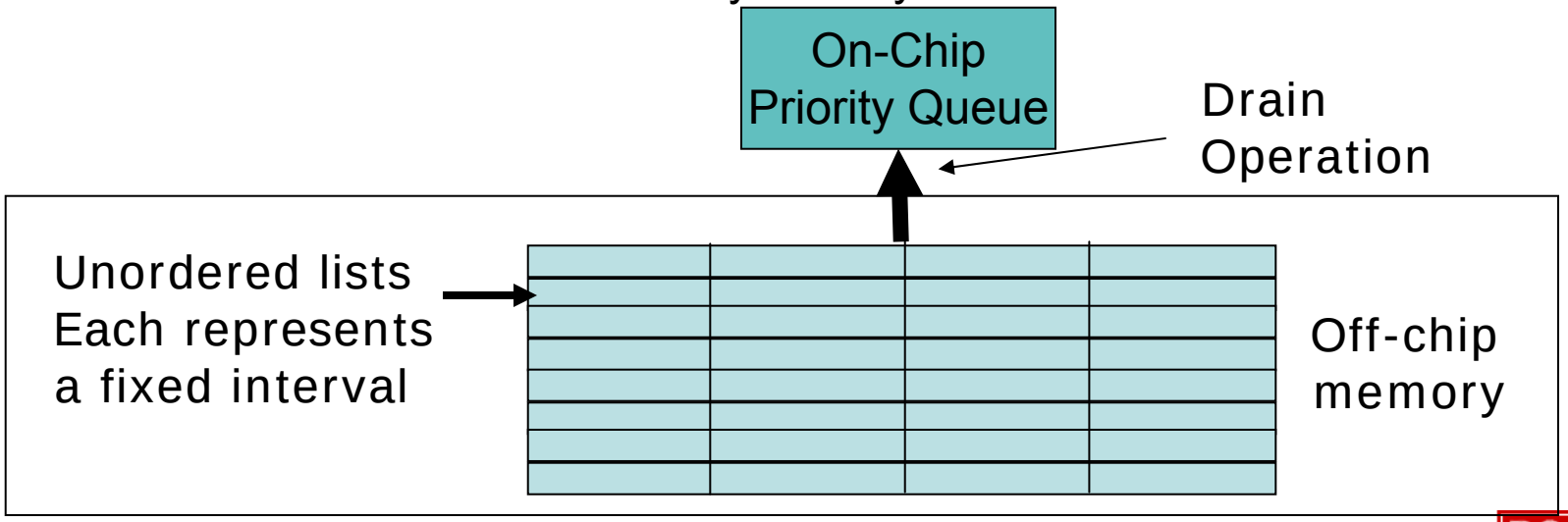
Advance queue – when priority queue is emptied, "drain" a list into a new one.

Invalidate event – follow pointer from bead memory. Remove from linked list



Off-chip Event Calendar

- Recall: must be able to queue, dequeue, and invalidate events – all with a throughput of 100Mhz
- Problem: Don't have bandwidth for following pointers!
- Sketch:
  - new events are appended to unordered lists – one list per time interval
  - lists are drained as they reach the head of the list queue
  - events are sorted as they are drained onto the FPGA
  - Events are checked for validity as they are drained



Off-chip Event Calendar – Processing

Dequeue next – not needed

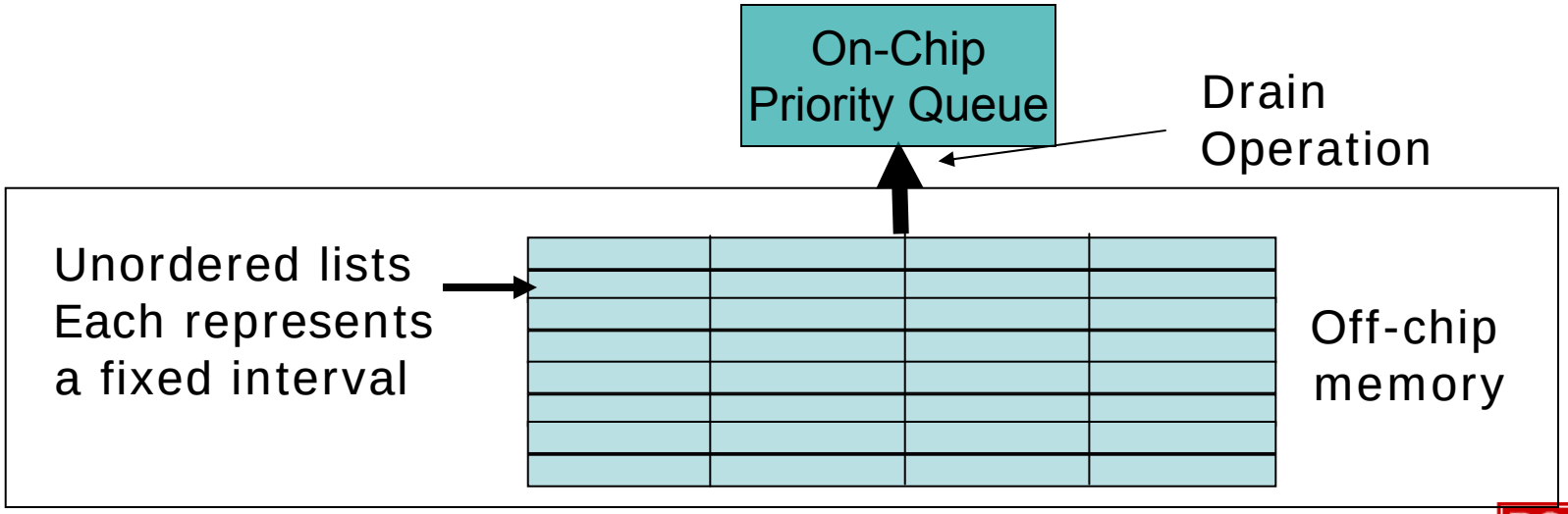
Insert – compute list as before. Each list is an array: append to end

Advance queue – stream next list into on-chip queue with insertion sort

Invalidate events – For each bead, keep track of

- Time of last invalidation
- Time at which the last event was queued

Check events as they are streamed onto the chip

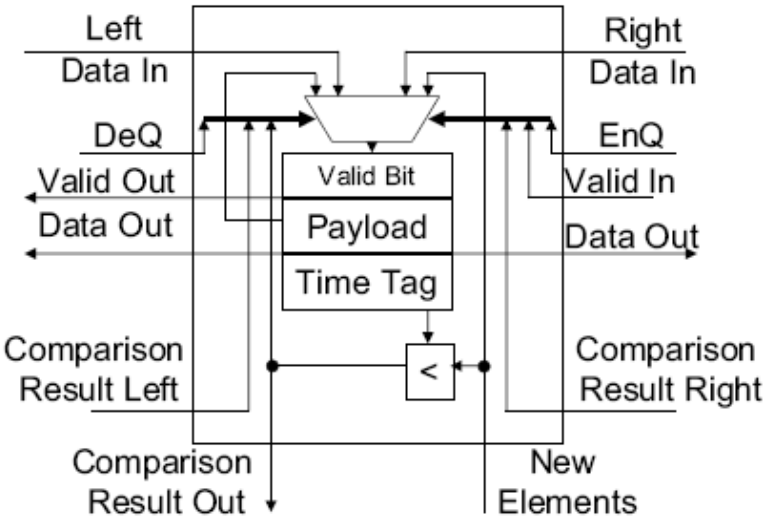


Outline

- Overview: MD, DMD, DES, PDES
- FGPA Accelerator Conceptual Design
- Design Complications – Dealing with …
- **FPGA Implementation and Performance**
- Multicore DMD
- Discussion

“Scrunching” Priority Queue Unit Cell Implementation

- Element Sizing
  - 32-bit tag
  - 26-bit Payload
  - 1-valid bit

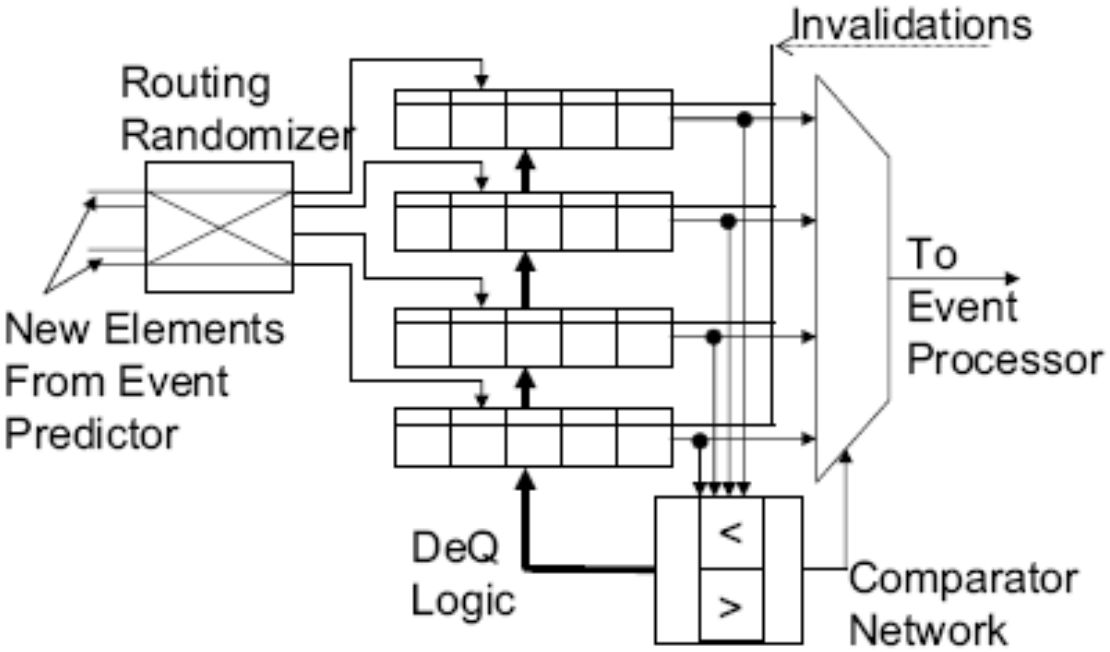


- Resources, 1000-stage
  - Xilinx V4, Synplify Pro, XST
  - 59059 Registers
  - 154152 LUTs

- Successfully constrained to 10ns Operation, post place-and-route

On-Chip, "Scrunching" Priority Queue

- 4 single insertion queues, and a randomizer network



Simulated Hardware Performance

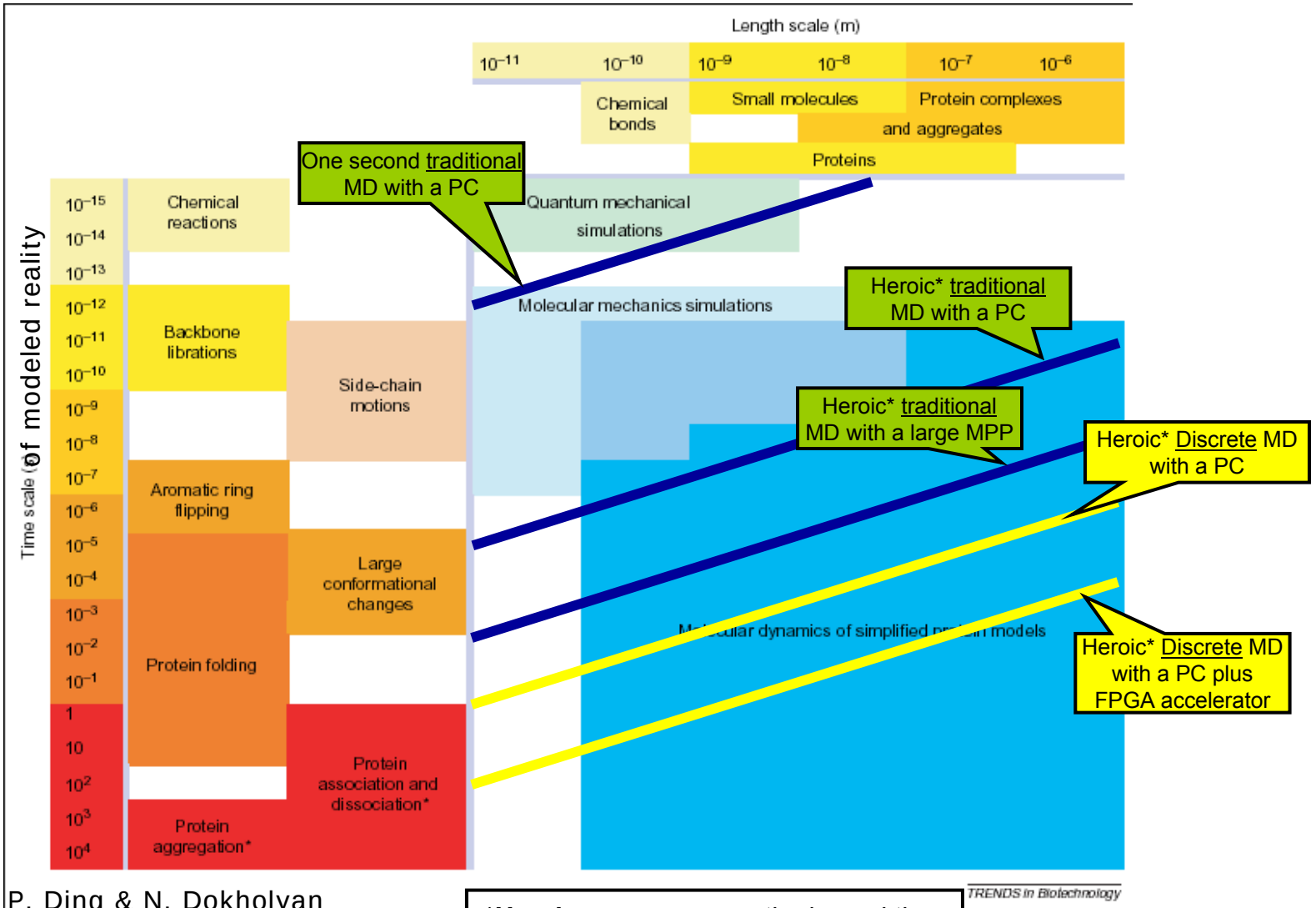
- Simulation parameters
  - 6000-Bead, Hard-sphere simulation
  - 32x32x32 Cell simulation box

- Two serial reference codes: Rapaport & Donev
- Two serial processors: 1.8GHz Opteron, 2GB RAM & 2.8GHz Xeon, 4GB RAM
  - Maximum performance achieved = 150 KEvents/Sec

- FPGA target platform: Xilinx Virtex-II VP70 w/ 6 on-board 32-bit SRAMs
- Operating frequency = 100Mhz
- Performance loss
  - Coherence - 2.1% of processed events → .48 stalls/commitment
  - Causality - 0.23% of processed events → .034 stalls/commitment
  - Scrunching – 99.9% events valid at commitment

- Overall, 65% of events are valid at commitment → **65 MEvents/Second**

DMD with FPGAs



P. Ding & N. Dokholyan
Trends in Biotechnology,2005

*__Heroic__ ≡ > one month elapsed time

Outline

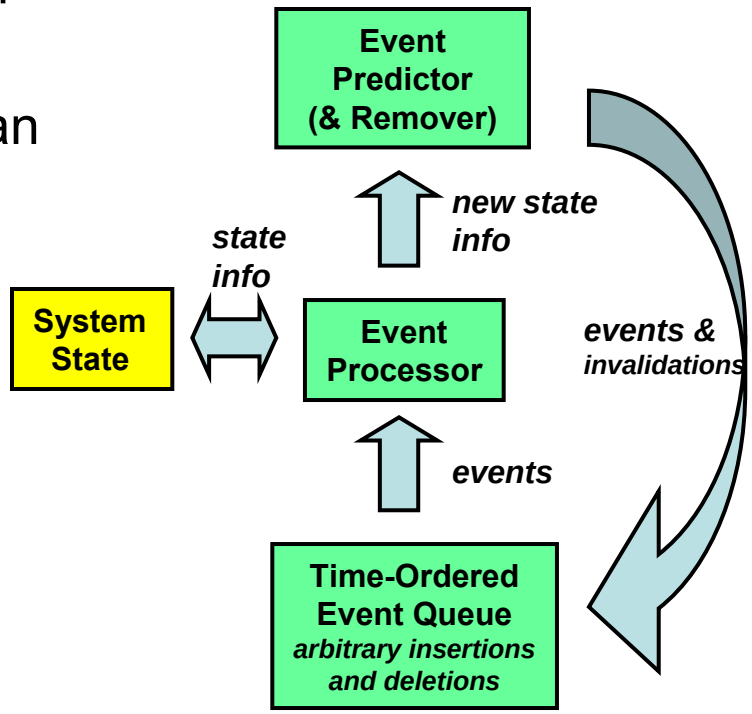
- Overview: MD, DMD, DES, PDES
- FGPA Accelerator Conceptual Design
- Design Complications – Dealing with …
- FPGA Implementation and Performance
- **Multicore DMD**
- Discussion

DMD Review

Parallelization requires dealing with hazards

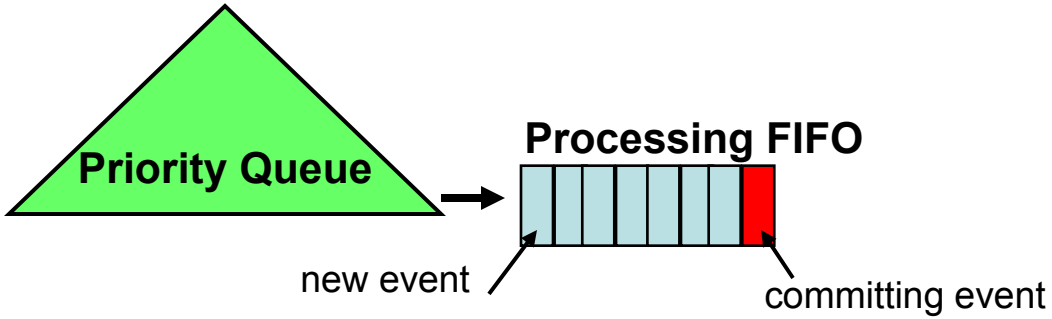
1. Causality – Out-of-order execution can lead to missed events
2. Coherence – Speculative prediction can lead to errors due use of to stale data

Approach – emulate FPGA event processing pipeline



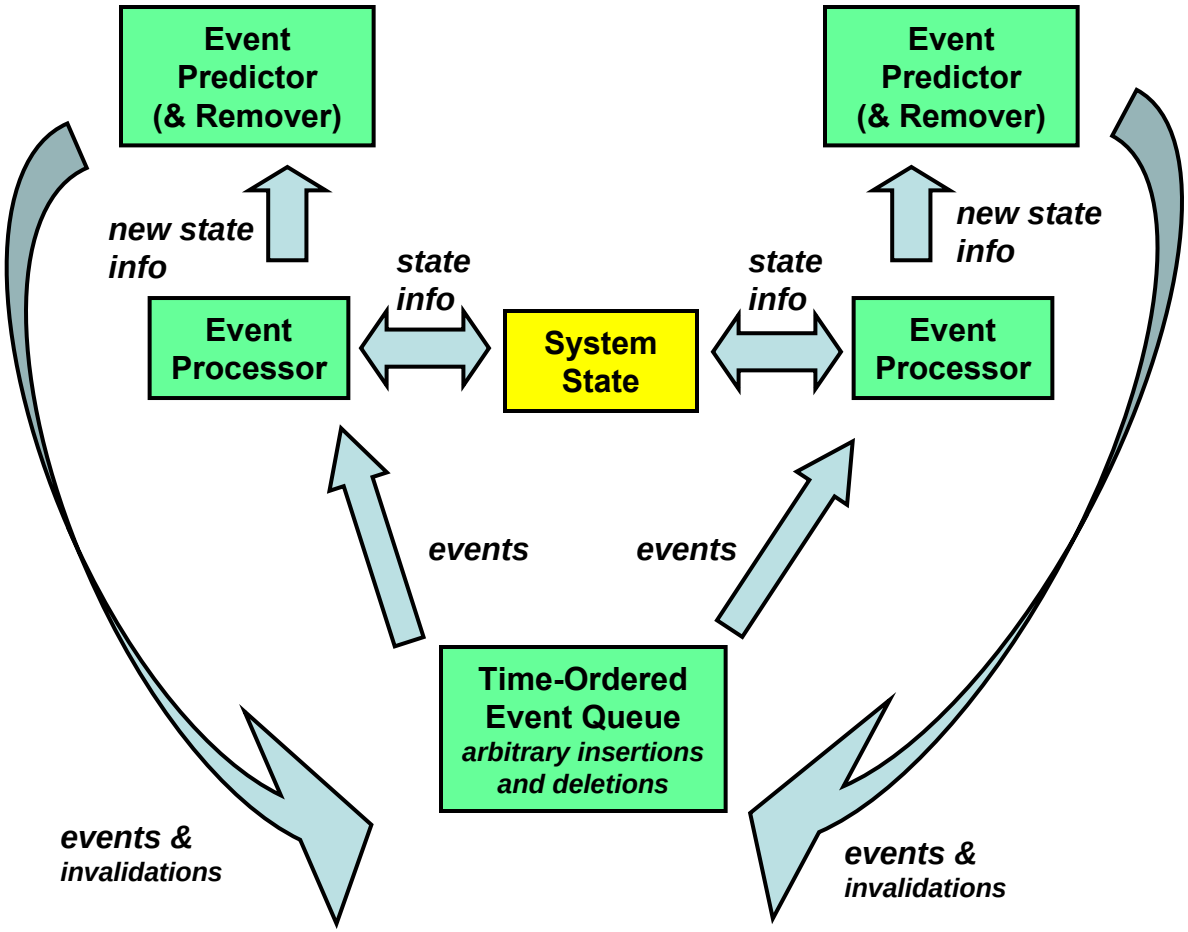
Multicore DMD Overview

- Task-based decomposition (task = event processing)
- Single event queue
- Several event executions in parallel
- Events committed serially and in order
  - Events dequeued for processing put into a FIFO

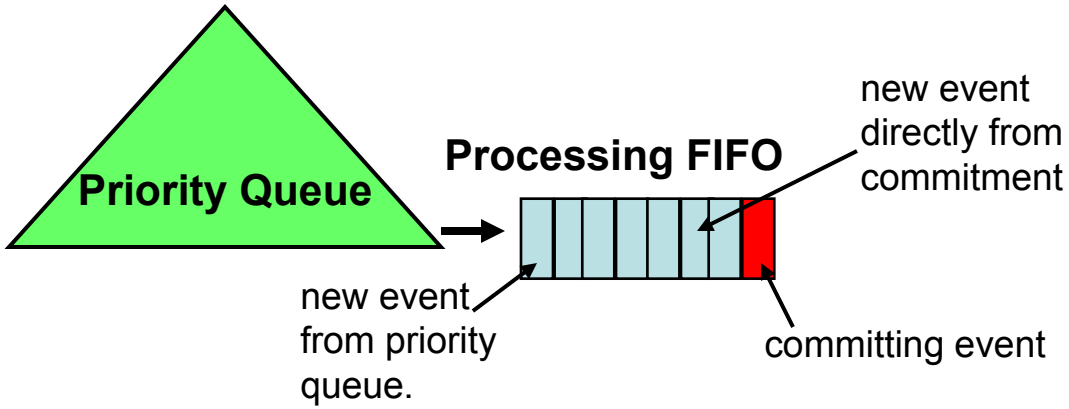


- Hazards must be handled in SW
  - Causality: insert new event into processing FIFO
  - Coherence: check neighborhood before prediction

DMD Task-Based Decomposition



Dealing with Hazards



1. Coherence →

- Events being enqueued in FIFO check "ahead" for neighborhood conflicts
- If conflict, then stall.

2. Causality →

- Newly predicted events can be inserted into correct FIFO slot

3. Causality + Coherence →

- Event inserted into FIFO must check "ahead" for coherence

4. Coherence + Causality →

- Events "behind" event inserted into FIFO must be checked for coherence
- If conflict, then restart

```
GetEvent
  WHILE (HoodSafe(EVENT) == FALSE)
    Check FIFO for EVENT(HoodSafe?) == FALSE  # check for orphans, but not 2nd time
    Check FIFO for EVENT(restart) == TRUE     # from "backwards" Hood checks
    Check TREE
      If TRUE then remove and append to FIFO
      ELSE drain a LIST
    # we now have an event
    Check for HoodSafe(EVENT)
      IF TRUE then EVENT(HoodSafe?) ← TRUE
      ELSE EVENT(HoodSafe?) ← FALSE

ProcessEvent
  Do event processing and prediction
  WAIT until head of FIFO

CommitEvent
  Update state                                  # Beads, Cells
  Remove EVENT from FIFO, put into FreeEventPool
  Invalidate EVENTs as needed
    follow from BEADs through all events in various structures
    Delete if in TREE or LISTS
    Cancel if in FIFO
  Insert new EVENTs
    get free EVENTs from FreeEventPool
    copy new data into EVENT structs
    update event structures
    for insertions into FIFO
      do Hoodcheck, set HoodSafe? as needed
      do Reverse hood check, set Restart as needed
```

Performance – Current Status

Experiment

Box Size = 32x32x32 cells
Particles = 131,000
Forces = Pauli exclusion only (hard spheres)
Particle types = 1
Density = .5
Simulation Models (of the simulation) = add processing delay to emulate processing of more complex force models

Multicore Platform =
2.5GHz Xeon E5420 Quad Core (1/08)

Threads	Model 1 0 delay	Model 2 delay = 46 us/event	Model 3 delay = 466 us/event
Baseline no thread support	6.04 us/event	52.8 us/event	472.3 us/event
1	0.81x	1.00x	1.00x
2	0.79x	1.64x	1.92x
3	0.47x	2.20x	2.80x
4	0.23x	2.39x	3.65x

Room For Improvement …

- Fine-grained locks
- Lock optimization
- Optimize data structures for shared access
- Change in event cancellation method (DMD technical issue)

Outline

- Overview: MD, DMD, DES, PDES
- FGPA Accelerator Conceptual Design
- Design Complications – Dealing with …
- FPGA Implementation and Performance
- Multicore DMD
- **Discussion**

Discussion

- Using dedicated microarchitecture implemented on an FPGA, very high speed-up can be achieved for DMD

- Multicore version is promising, but requires careful optimization

Future Work

- Integration of off-chip priority queue
- Predictor network
- Inelastic collisions and more complex force models
- Hydrogen bonds
- Explicit solvent modeling

Questions?