Explicit Communication Architectures for High-End Computing

Bill Dally
Computer Systems Laboratory
Stanford University
April 20 2005

The Problem: Sustained perf/$

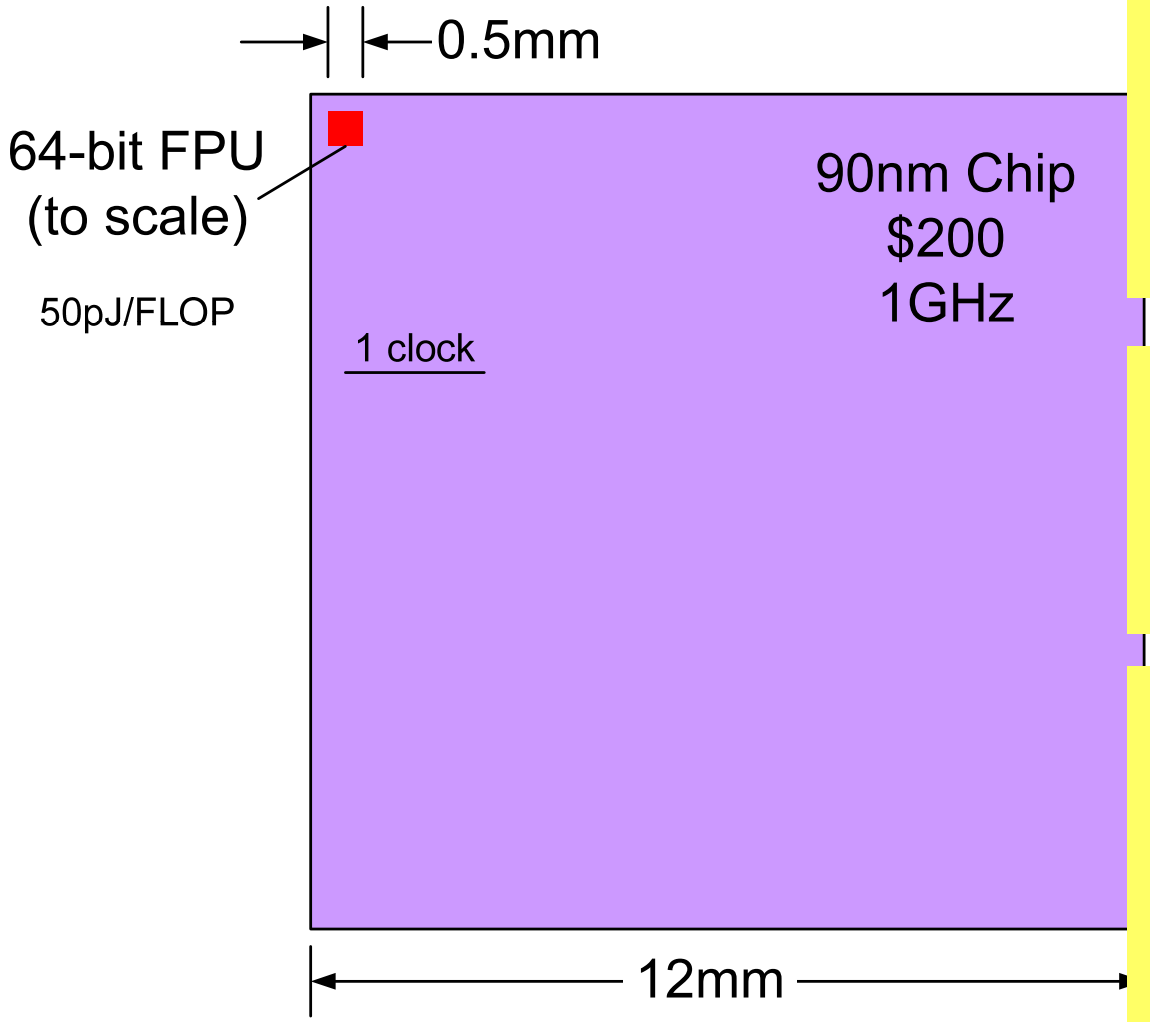
- Capability: Maximum sustained performance for X dollars
  - (X ~ $200M)
- Capacity: Maximum sustained performance per dollar

- Either way goal is “Sustained performance per dollar”
  - Only difference is scalability (which you have to pay for)

- Sustained performance is blend of performance on
  - Compute-limited part (FLOPS)
  - Local memory bandwidth limited part (GB/s – local)
  - Global bandwidth limited part (GB/s – global)

- Different elements have different sensitivity to cost

Technology makes arithmetic cheap and bandwidth expensive



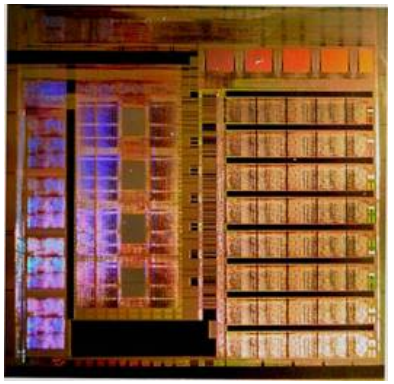
100s of FPUs per chip
$0.50/GFLOPS
50mW/GFLOPS

40GB/s off-chip BW
$5/GB/s
0.25W/GB/s

Cost of BW increases with distance
4x over backplane
6x over cable
25x VSR optics

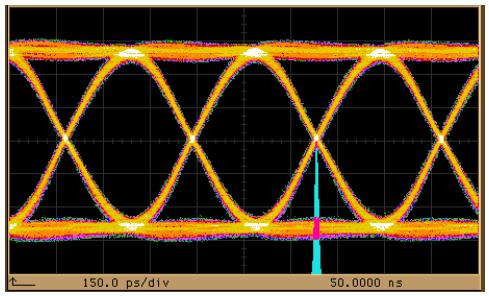
Cost is dominated by bandwidth (and memory)

- Arithmetic is cheap $0.50/GFLOPS,
  - (200GFLOPS chips)



- Memory is $200/GByte, ~$10/GB/s
  - 1GByte of memory costs 400GFLOPS
  - 1GB/s of bandwidth costs 20GFLOPS

- Global bandwidth moderate cost
  - $1 (board), $4 (backplane), $25 (fiber) per GB/s
  - 2GFLOPS (board), 8GFLOPS (backplane), 50GFLOPS (global)



Bandwidth is the critical issue, not FLOPS

- Bandwidth drives cost
  - 1WPS of memory BW = 160 FLOPS
  - 1WPS of global BW = 800FLOPS (2 fiber hops)
  - These ratios are getting larger over time

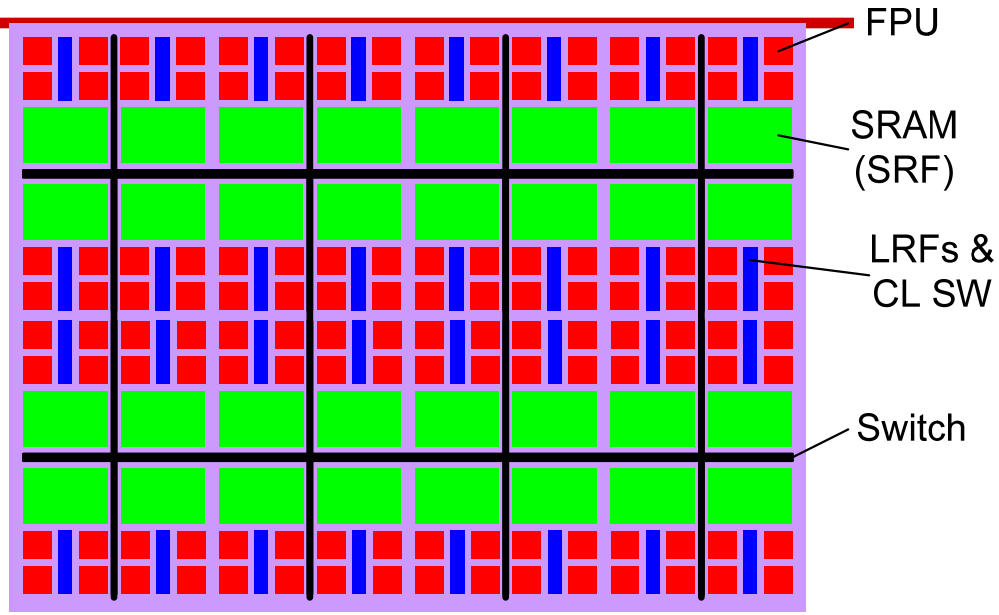
- Goal is to make efficient use of this costly, scarce resource
  - Keep it busy
    - Latency hiding
      - 500 words in flight today 1000s in near future
    - Overprovision arithmetic
      - To keep expensive BW occupied
  - Use it efficiently
    - Transfer only needed data (short cache lines)
    - Avoid transfers where possible (locality)

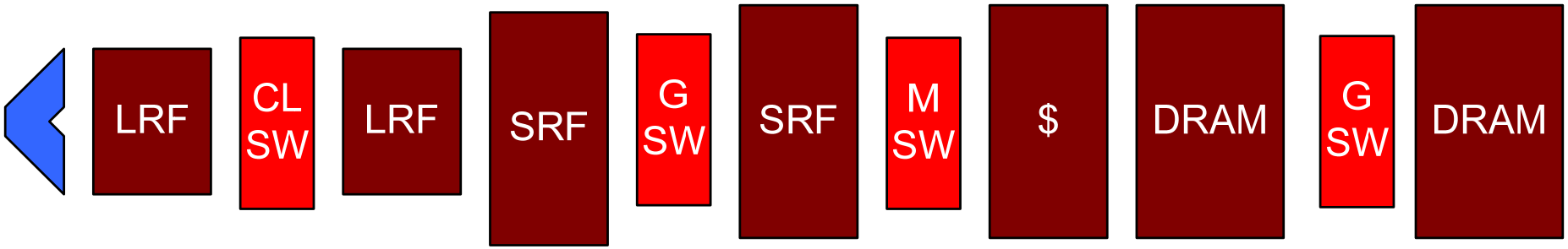
Exposed Communication

- Bandwidth is the critical resource
  - Make its use visible
  - Enable optimization by programmer and compiler
- Exploit producer-consumer locality
- Predictable and controllable storage – enables compiler
- Hides latency – with precision
- Enables more FLOPS per chip (per unit BW)

Register Hierarchy

- To expose communication, make storage explicit
- Communication takes place both between levels and within a level



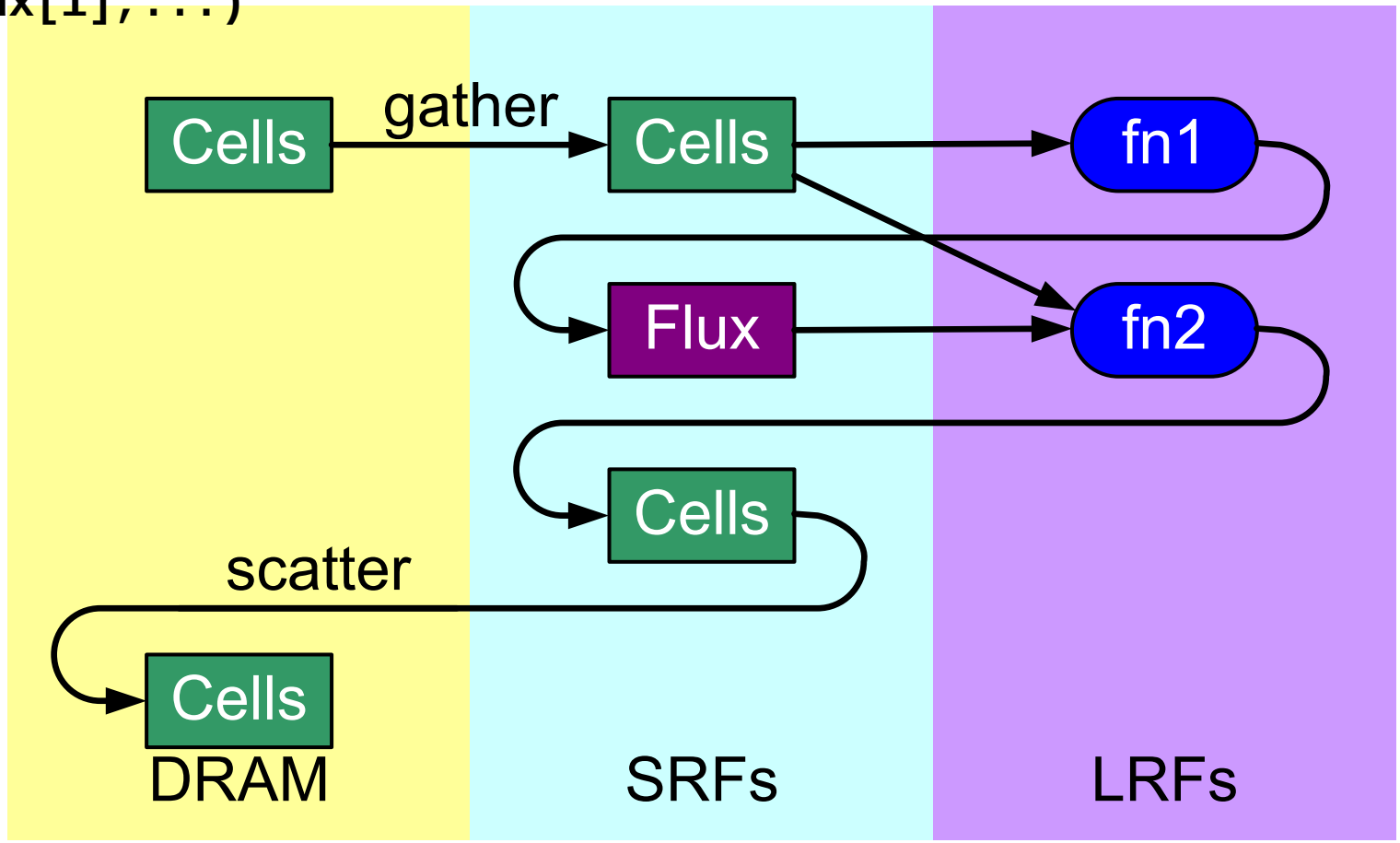


Producer-Consumer Locality

```
loop over cells
  ...
  flux[i] = ...
  ...

loop over cells
  ...
  ... = f(flux[i],...)
```

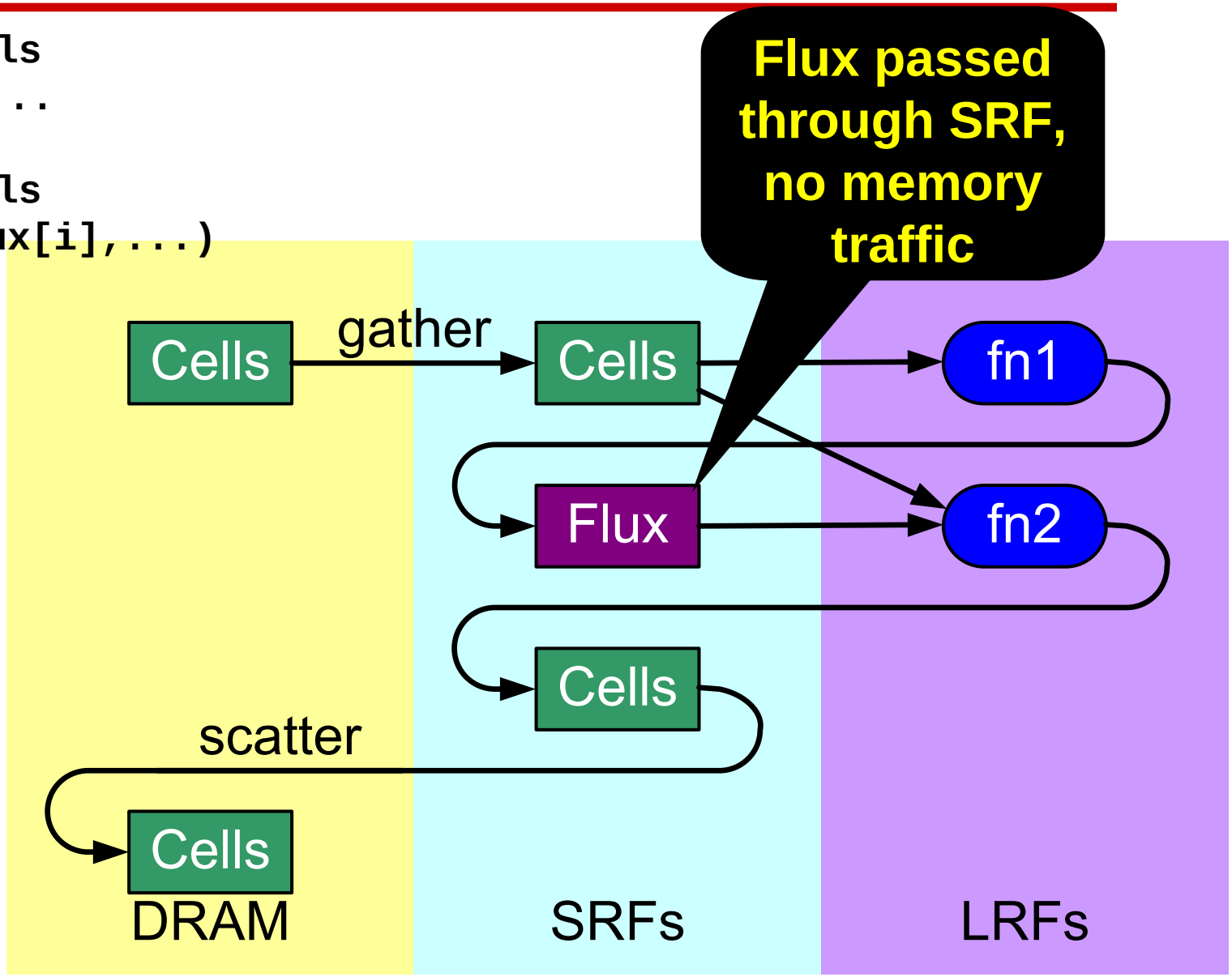
Explicitly block into SRF

```
loop over cells
   flux[i] = ...

loop over cells
   ... = f(flux[i],...)
```



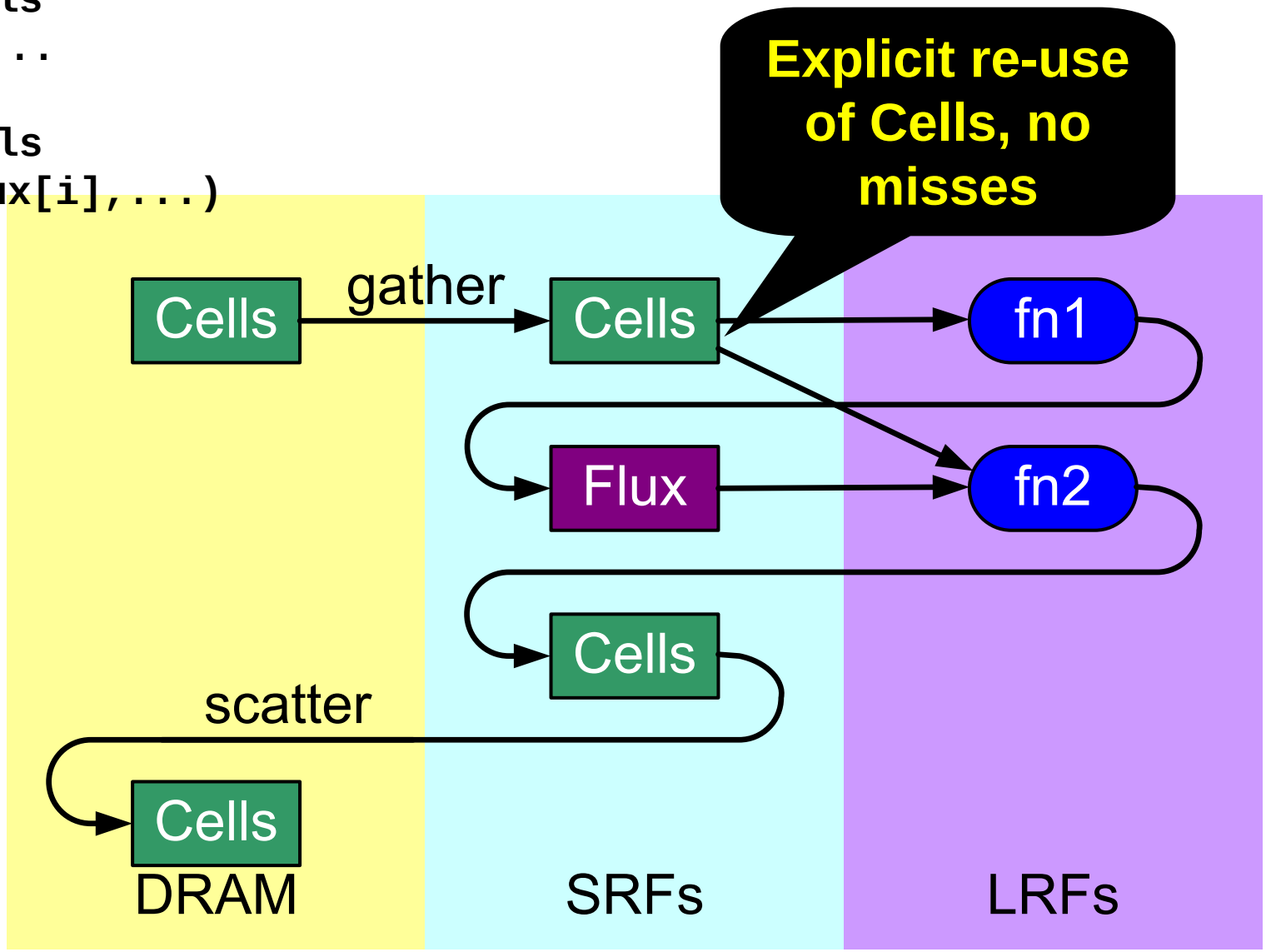
Explicitly block into SRF

```
loop over cells
   flux[i] = ...

loop over cells
   ... = f(flux[i],...)
```

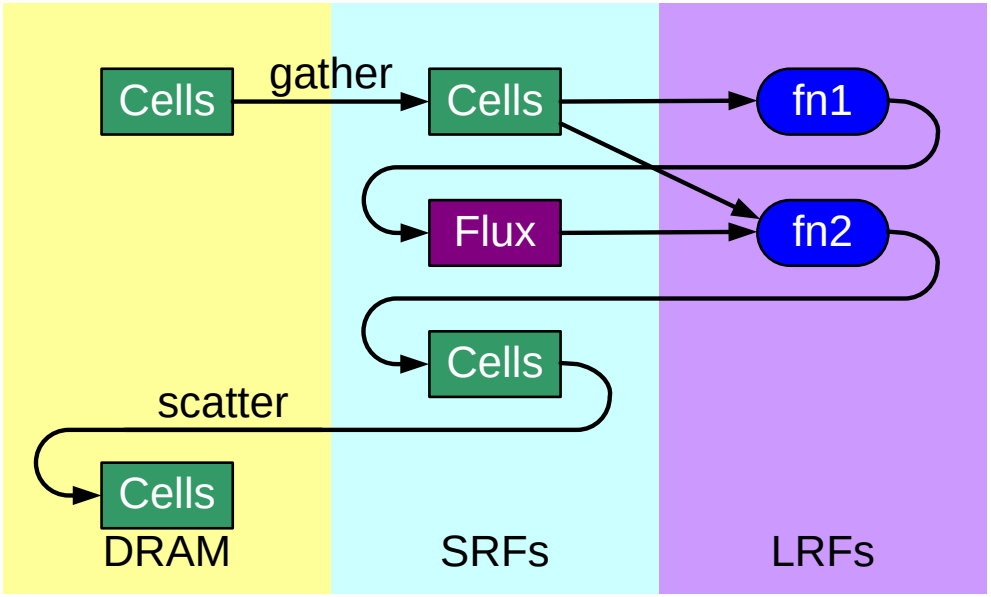


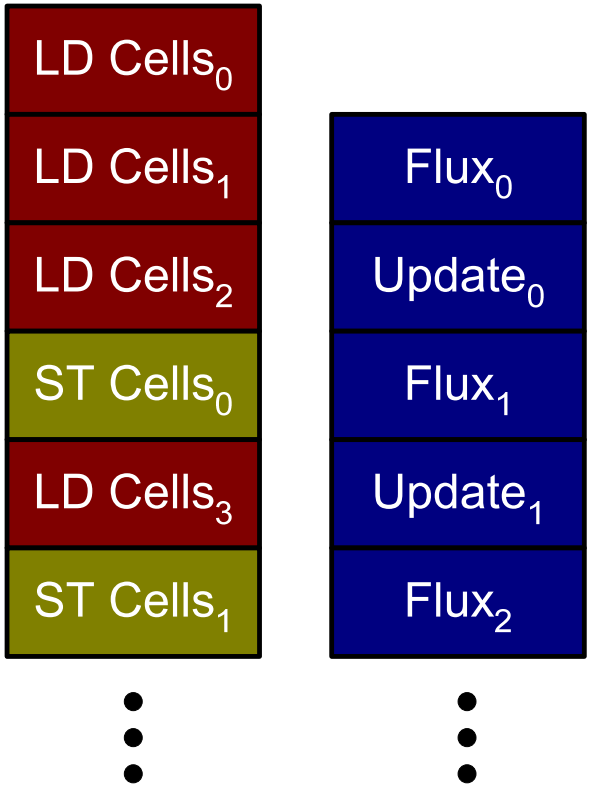
Explicitly block into SRF

```
loop over cells
   flux[i] = ...

loop over cells
   ... = f(flux[i],...)
```

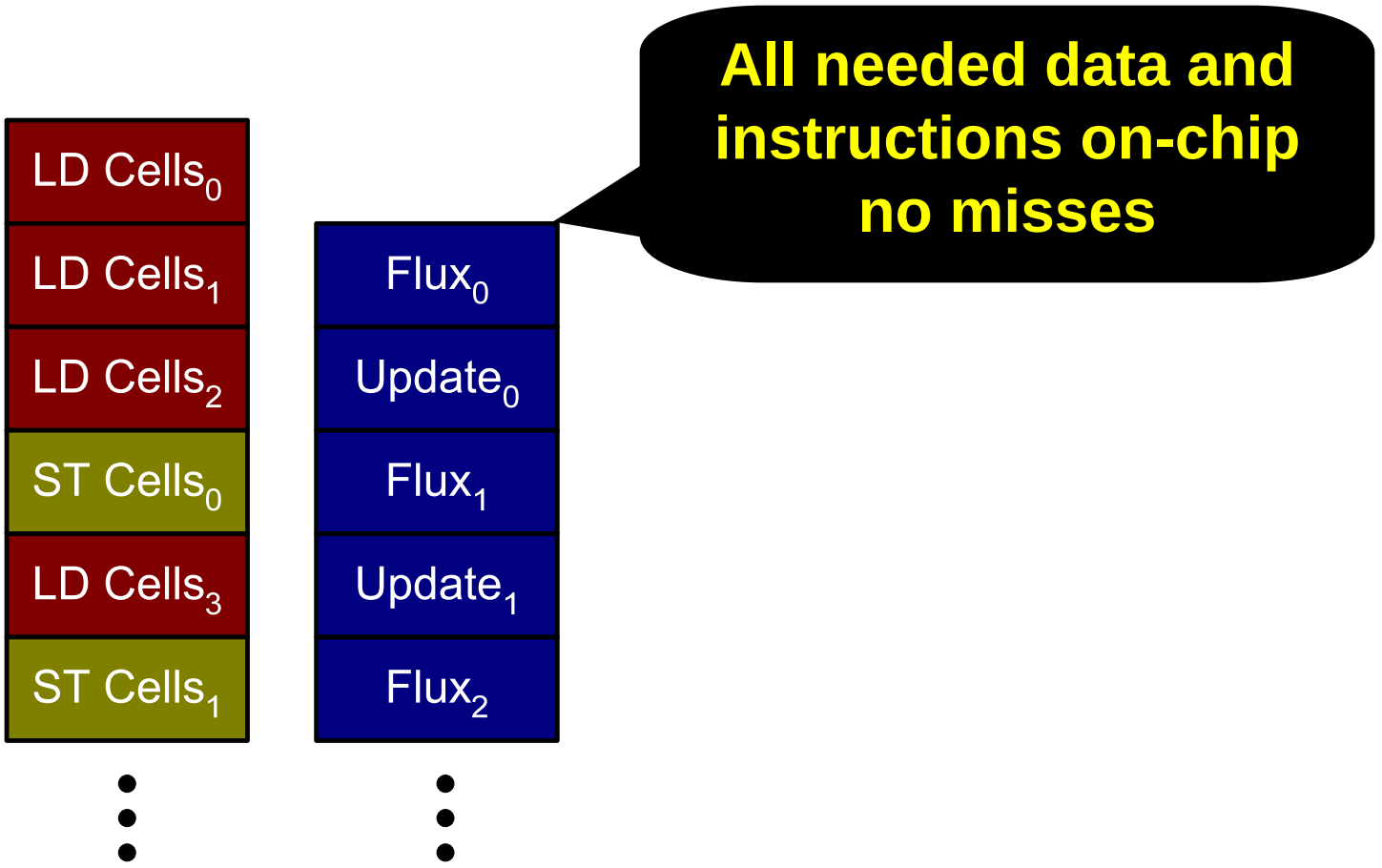


Stream loads/stores hide latency (1000s of words in flight)

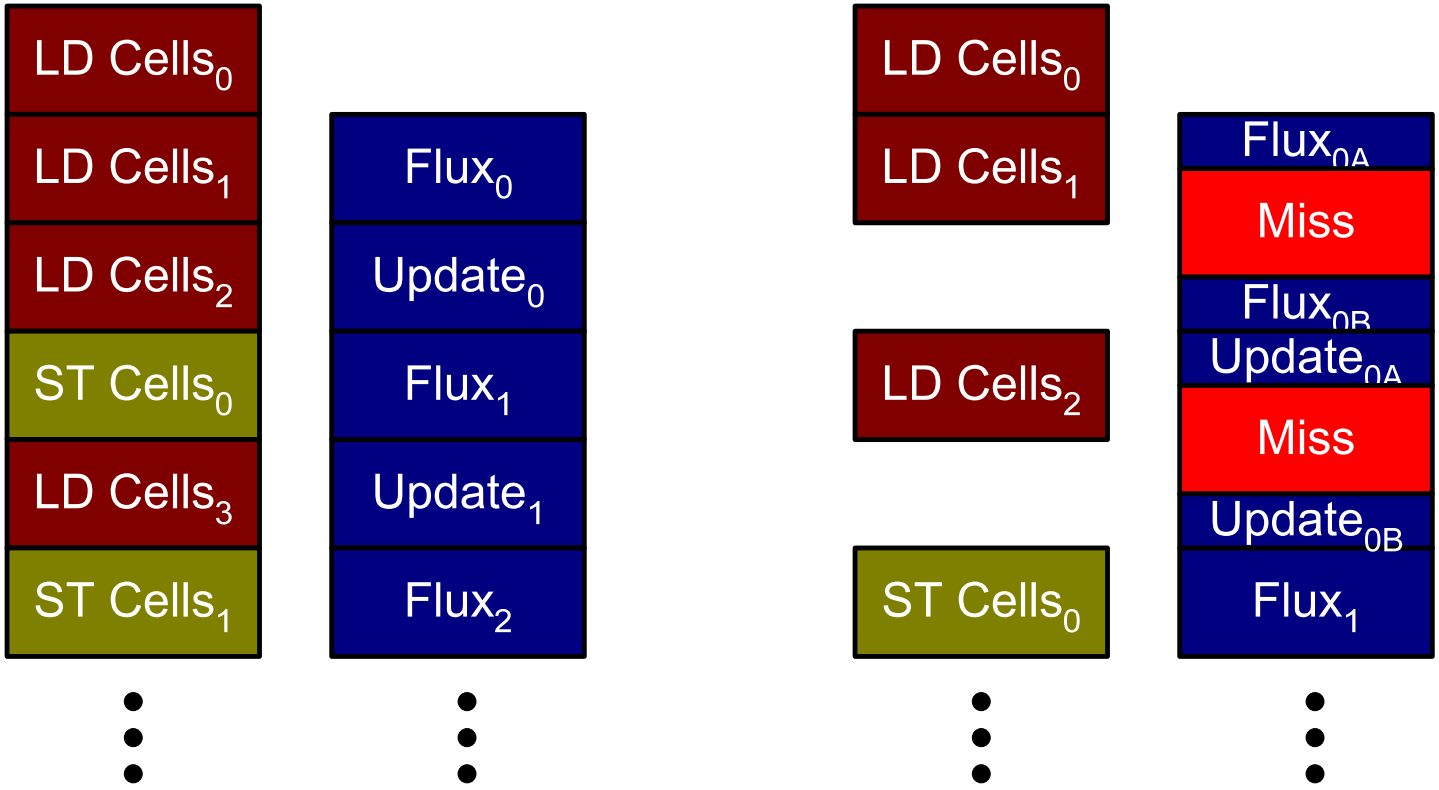




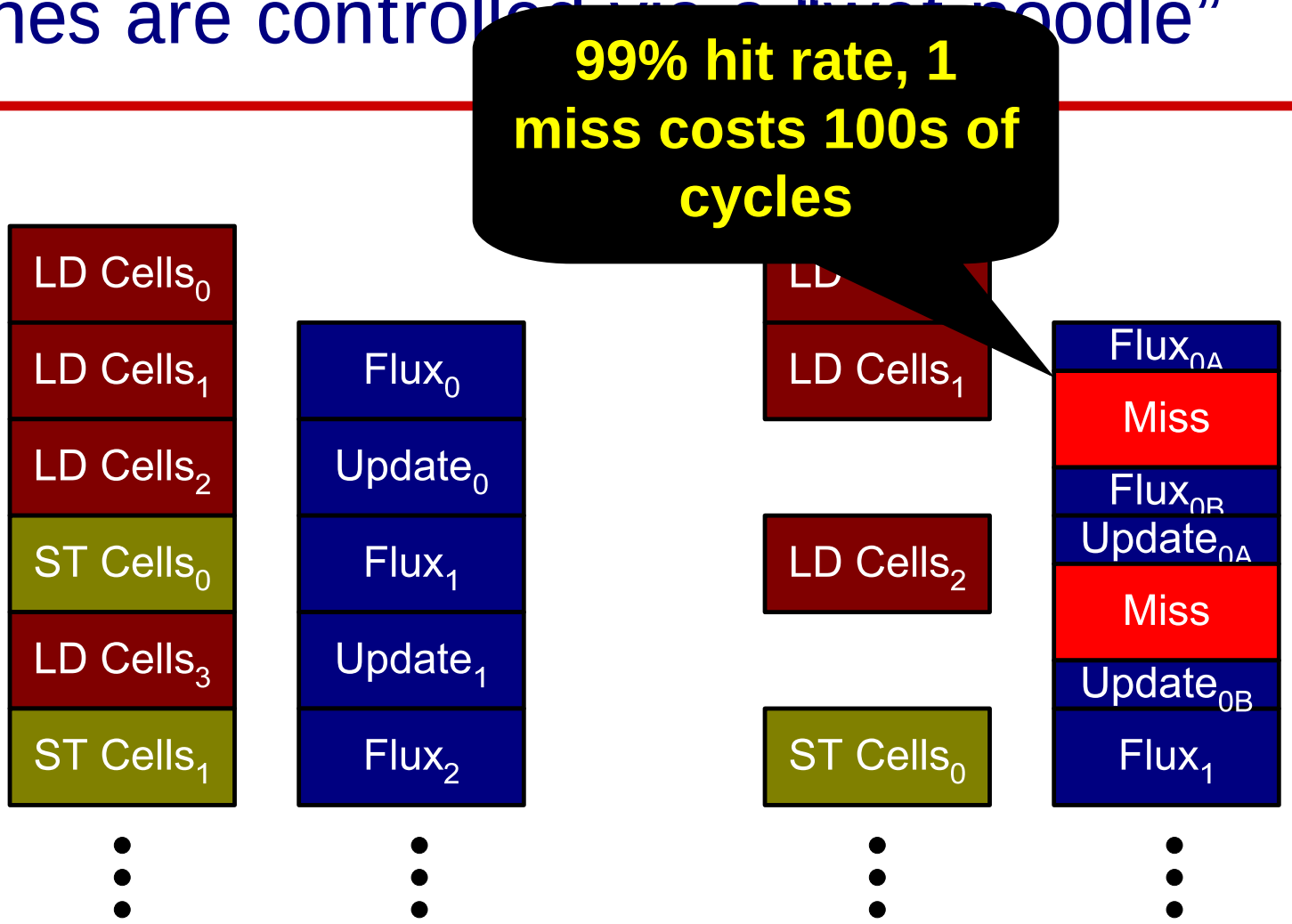
Explicit storage enables simple, efficient execution



Caches are controlled via a “wet noodle”



Caches are controlled via a "wet noodle"



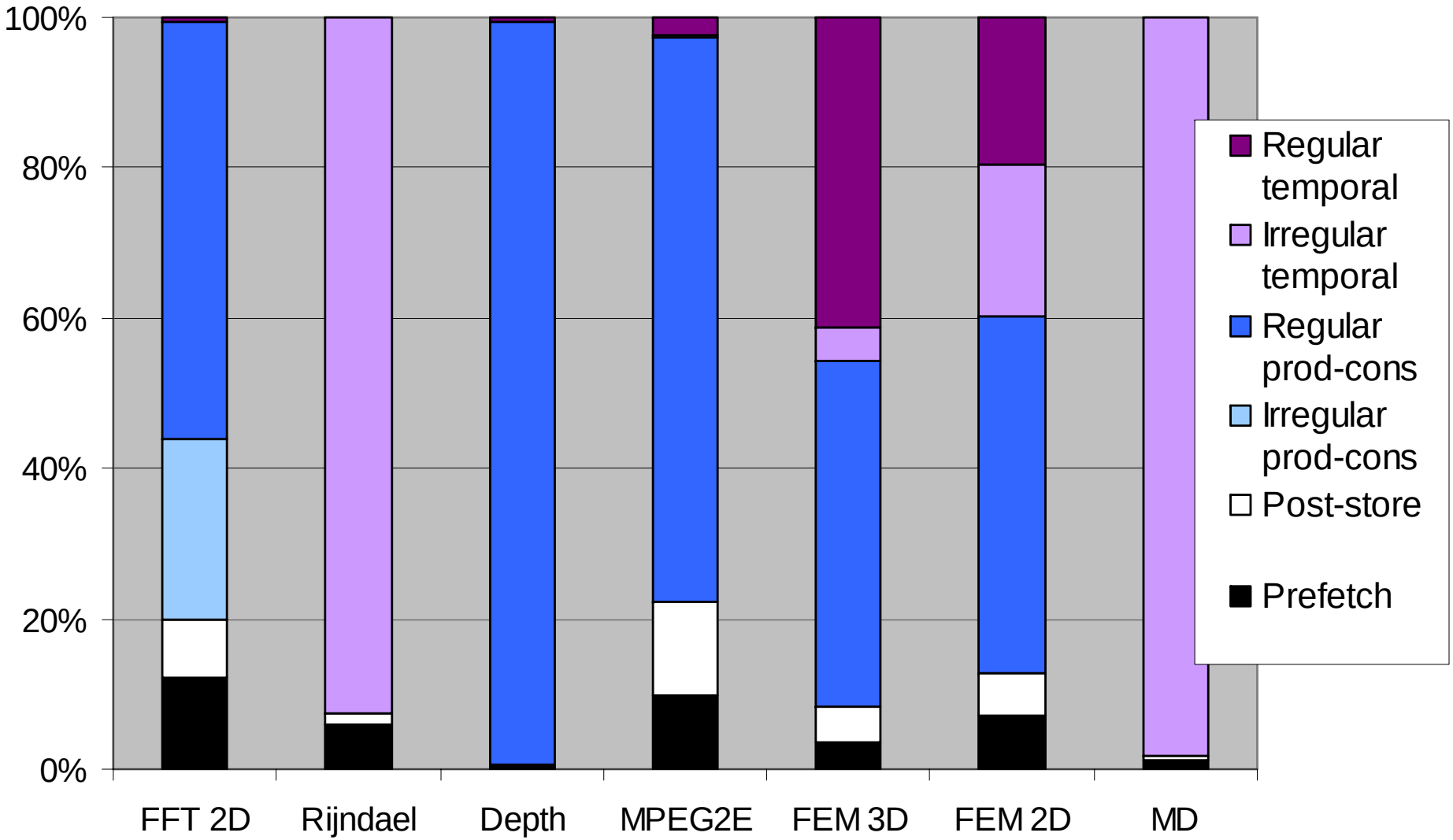
Explicit storage vs. Cache

- All data and instructions local before starting work
  - vs. periodic misses with high penalties
  - No unexpected conflict/capacity misses
- Only needed data loaded
  - vs. full cache line
  - vs. read on allocate
- No traffic consumed for dead data
  - vs. writeback of all dirty data (dead or alive)

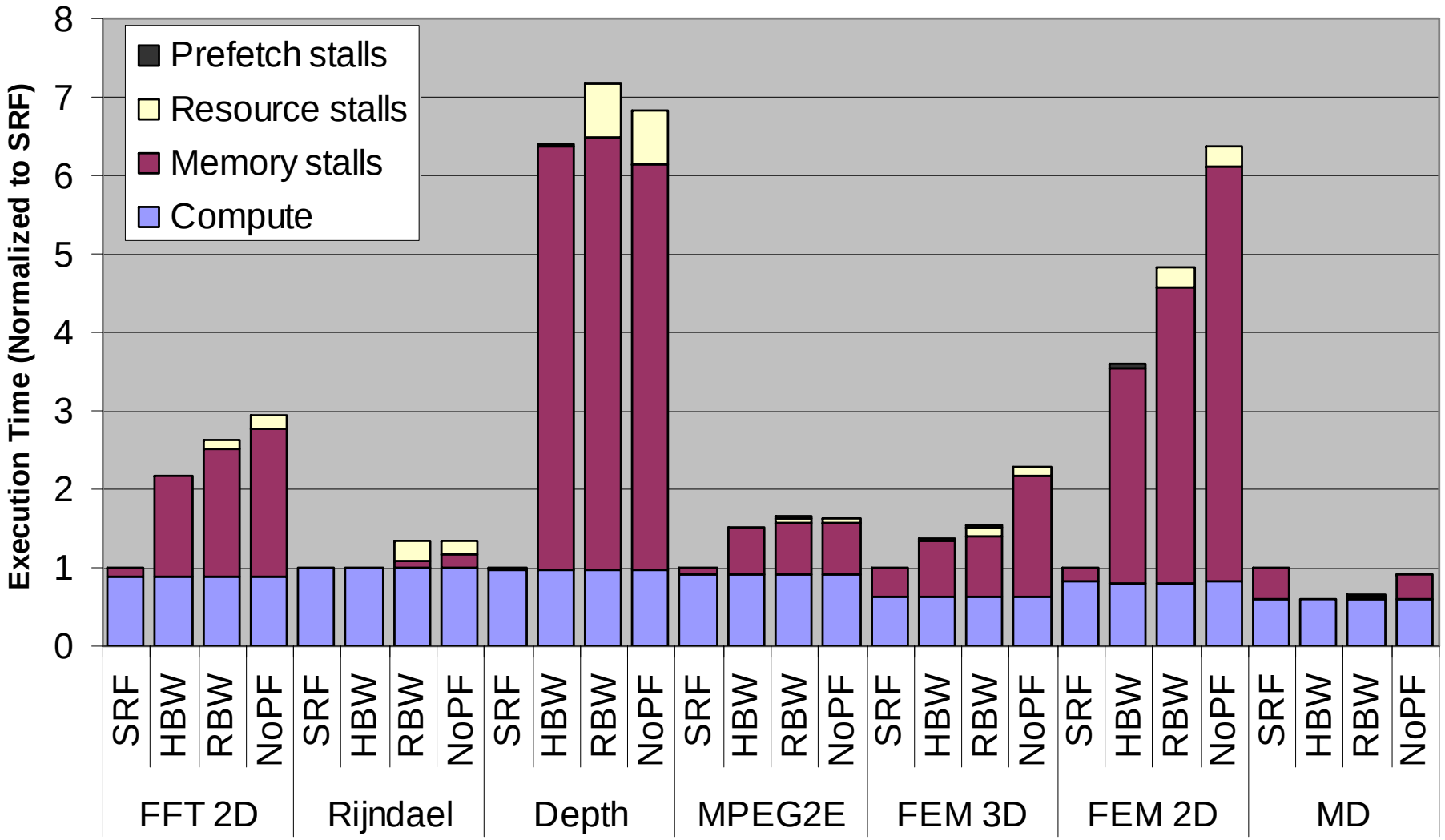
Explicit Storage vs. Vectors

- Similar concept at a larger scale
- Records vs. words
  - Larger burst size in DRAMs
- Transfer 1,000 – 10,000 words per reference
  - vs. 64-128
  - Able to hide Latency x Bandwidth (500 today and growing)
- Vector registers ~ LRFs
  - SRF is new (and needed) level of hierarchy

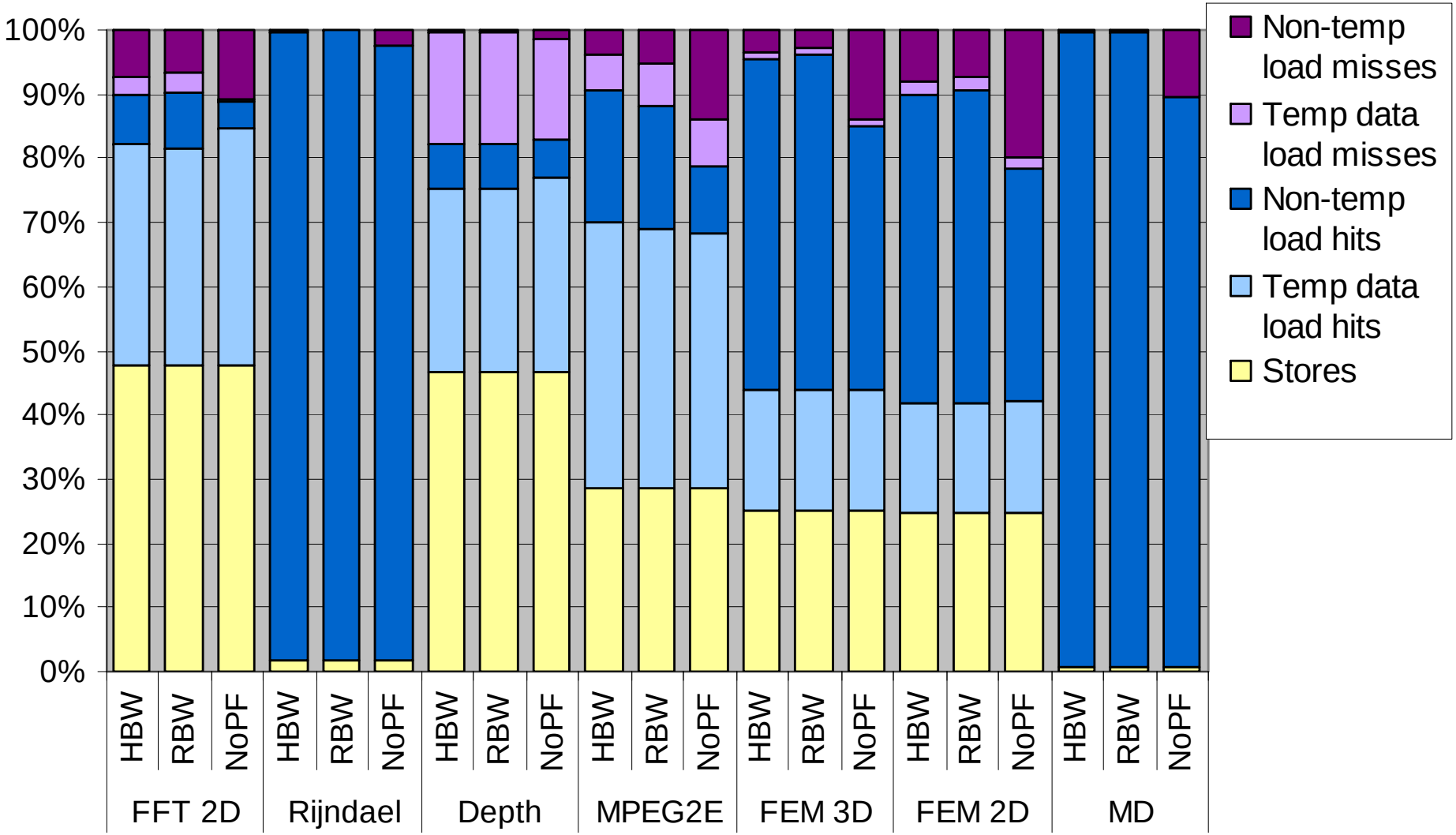
Benchmark Memory Usage



Execution Times



Cache Miss Behavior



Locality – requires both HW and SW

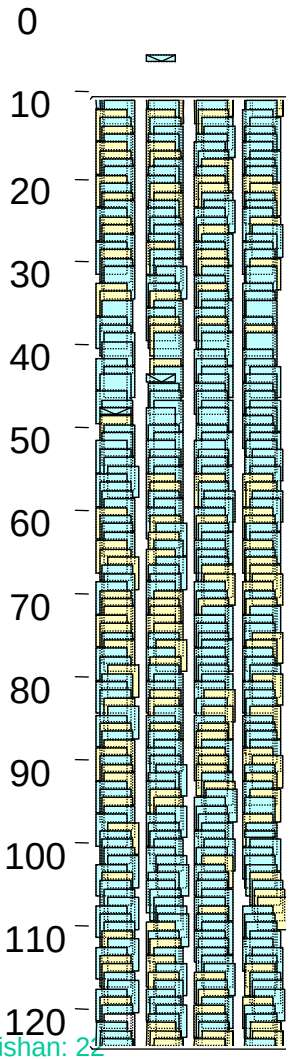
- Hardware provides explicit storage hierarchy
- Software maps objects to this hierarchy to minimize bandwidth
  - Can't do the SW without the HW

```
subXFlux(…) {
  loop over elements
    compute X flux

subYFlux(…) {
  loop over elements
    compute Y flux
```

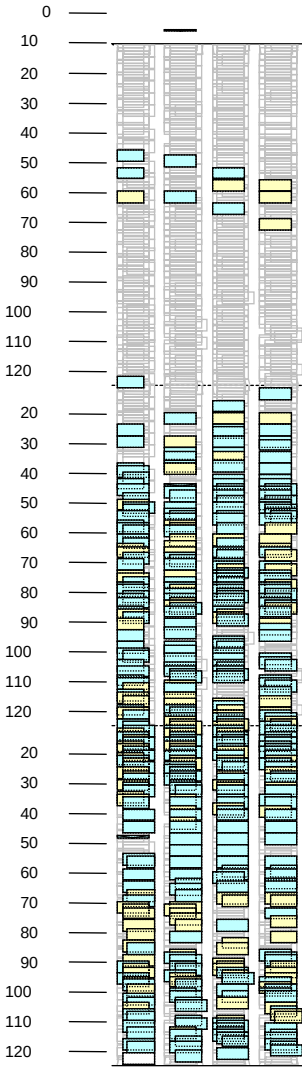
```
subXYFlux(…) {
  loop over elements
    compute X flux
    compute Y flux
```

Explicit storage enables simple, efficient execution unit scheduling

One iteration



SW Pipeline

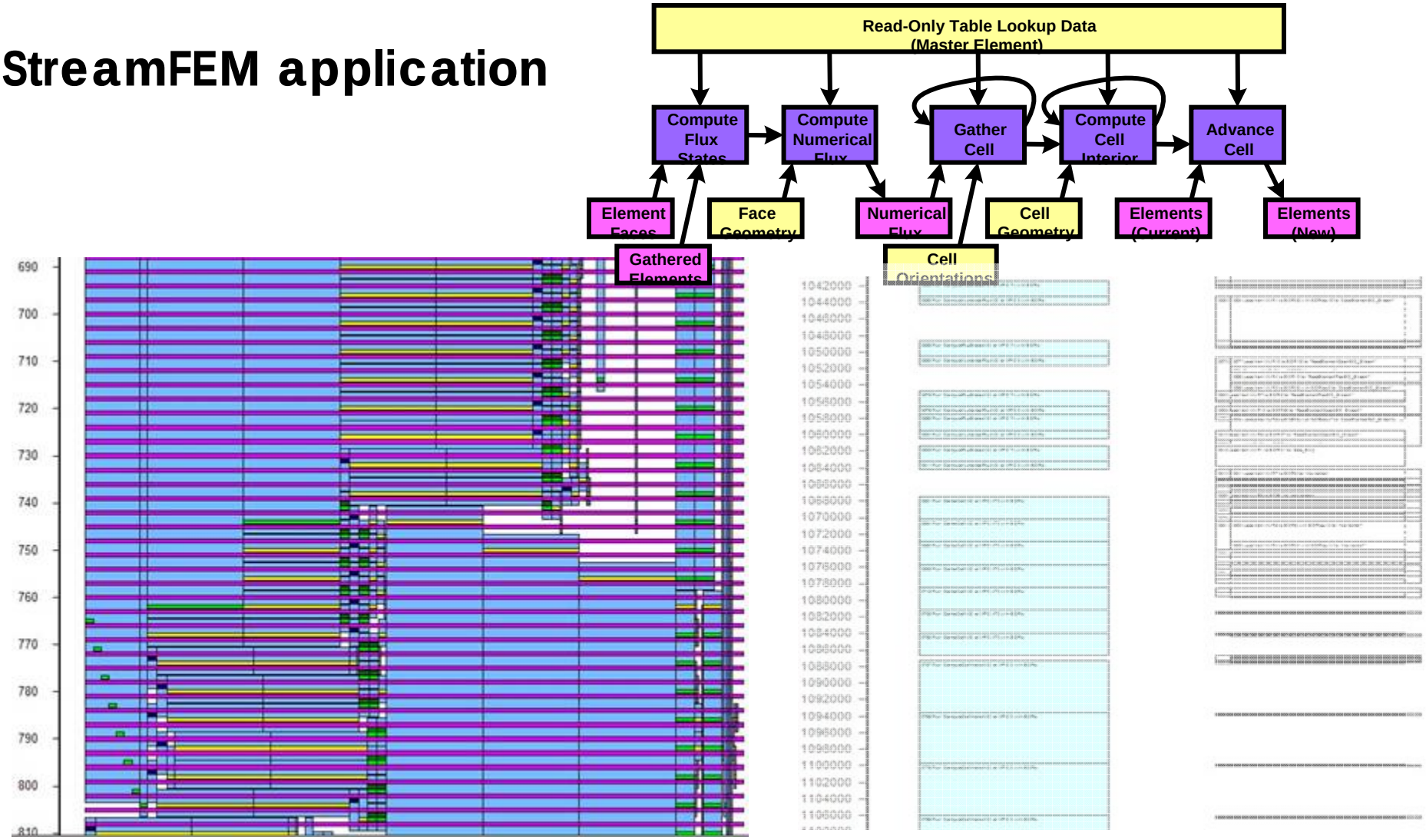


ComputeCellInt kernel from StreamFem3D

Over 95% of peak with simple hardware

Stream scheduling exploits explicit storage to reduce bandwidth demand

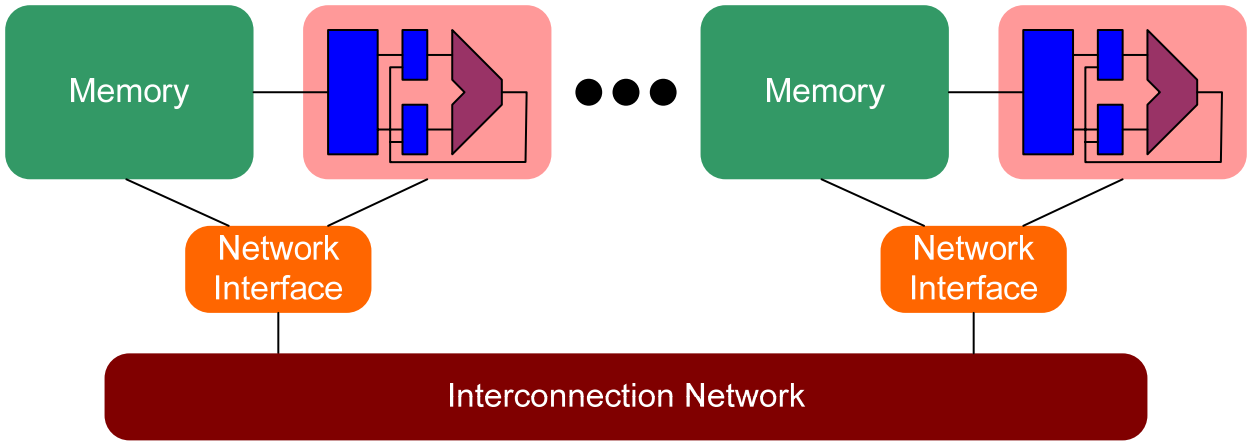
StreamFEM application



Prefetching, reuse, use/def, limited spilling

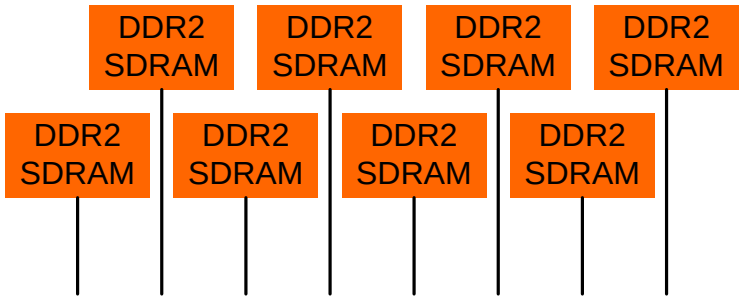
Bandwidth- (and memory-) centric architecture
A recipe

- Provide most economical memory bandwidth and capacity
  - Commodity DRAM chips (DDR-2 or GDDR or XDR)
- Need chips to connect to these DRAMs
  - Fill these chips with
    - 64-b FPUs – 100s - overprovision for compute-limited parts
    - Local storage – to reduce demand on bandwidth
- Connect these chips together with an efficient network
  - High-radix routers
  - High-speed signaling



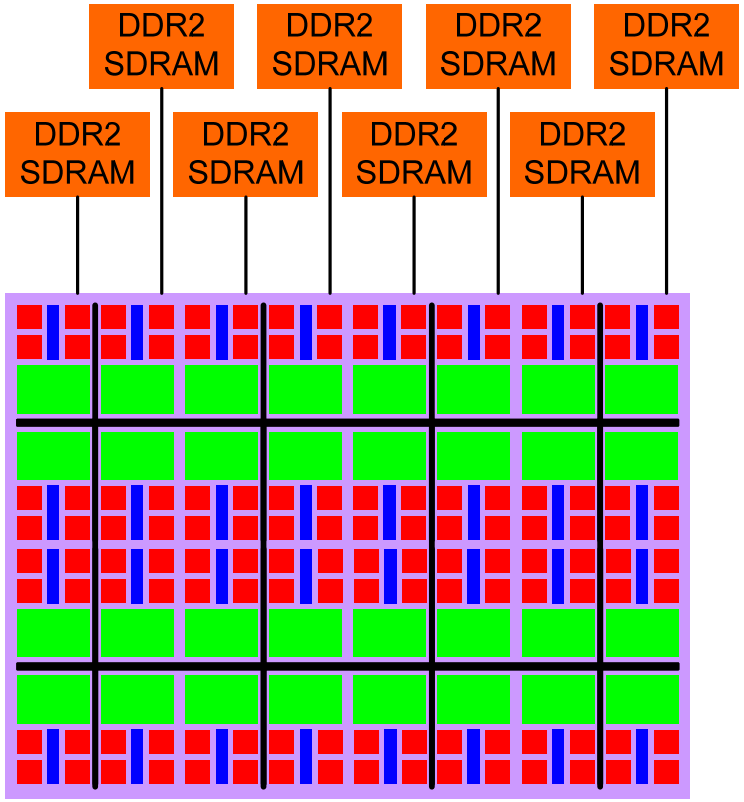
First provision memory (capacity & bandwidth)

- Commodity DRAM
  - $200/GByte,
  - $10/GByte/s
- No pin multiplexing



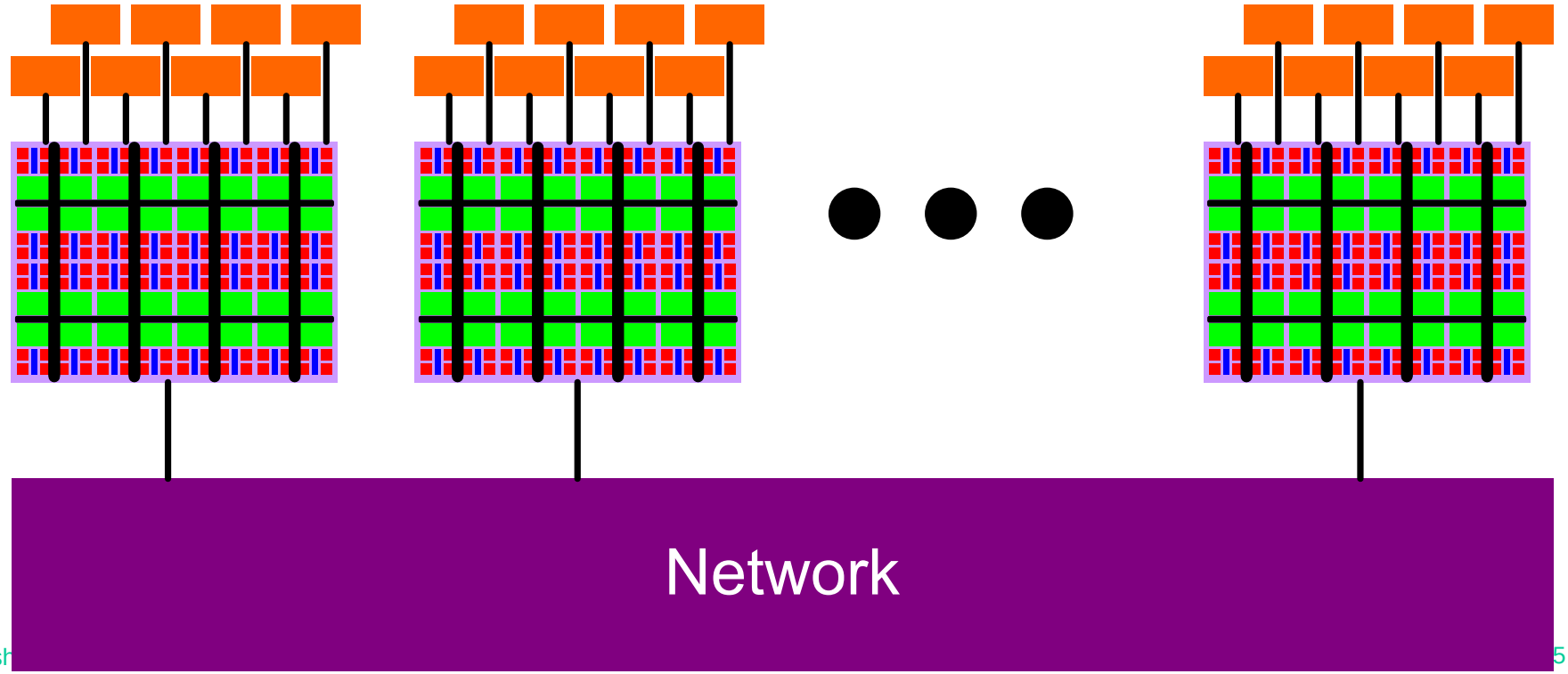
Fill memory interface chip with FPUs regs and local memory (and switches)

- Attach as much memory as one chip can handle with no pin multiplexing
  - 2GBytes
  - 40Gbytes/sec
- $400 of memory
- $200 Chip
- Fill chip with FPUs and explicit storage hierarchy

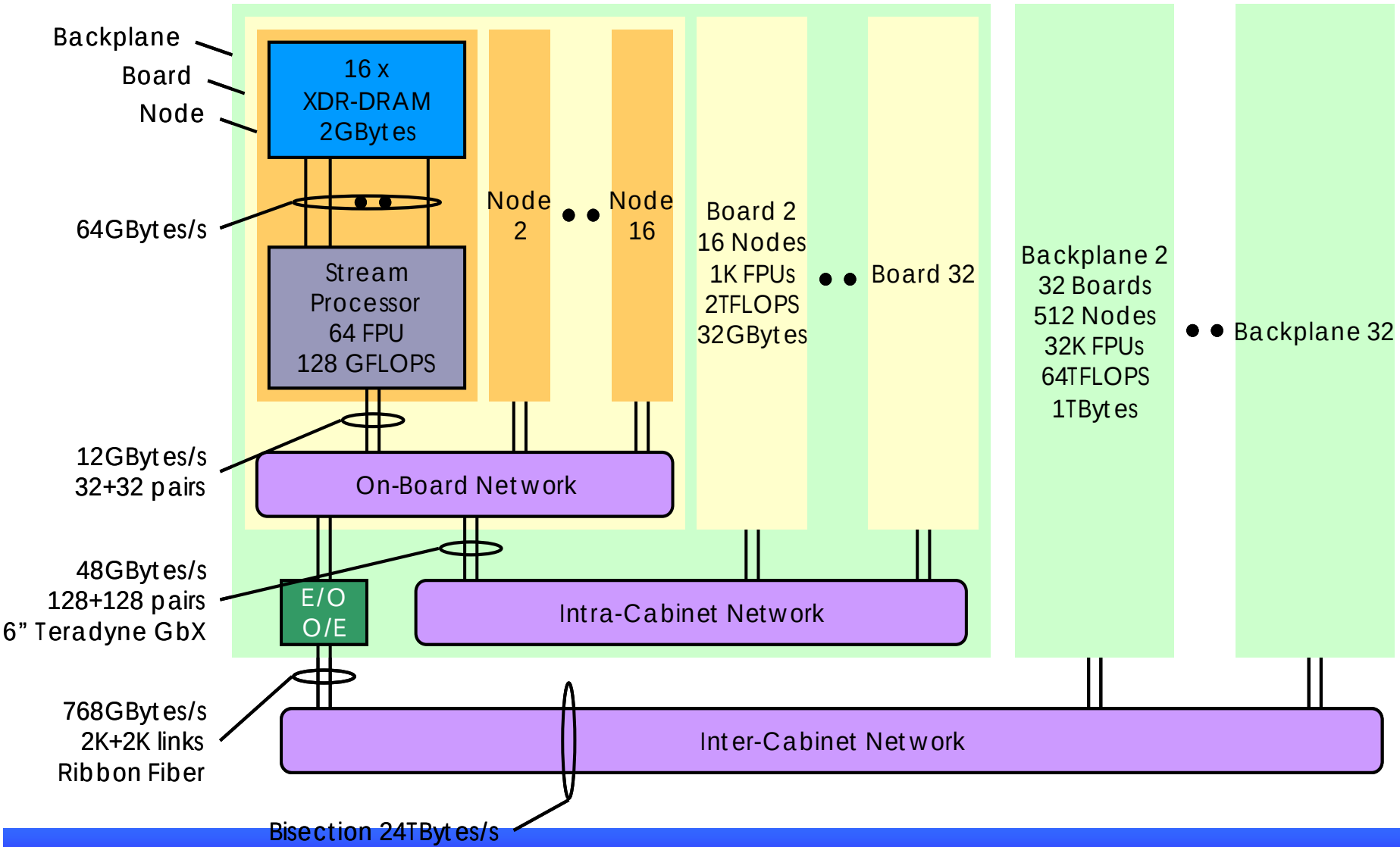


Connect these nodes together with an efficient network

- Bandwidth taper driven by cost
- Flat on PCB
- 4:1 in cabinet
- 8:1 across system



Merrimac – Streaming Supercomputer



Scalable from 2-TFLOP workstation to 2-PFLOP supercomputer

Merrimac Application Results

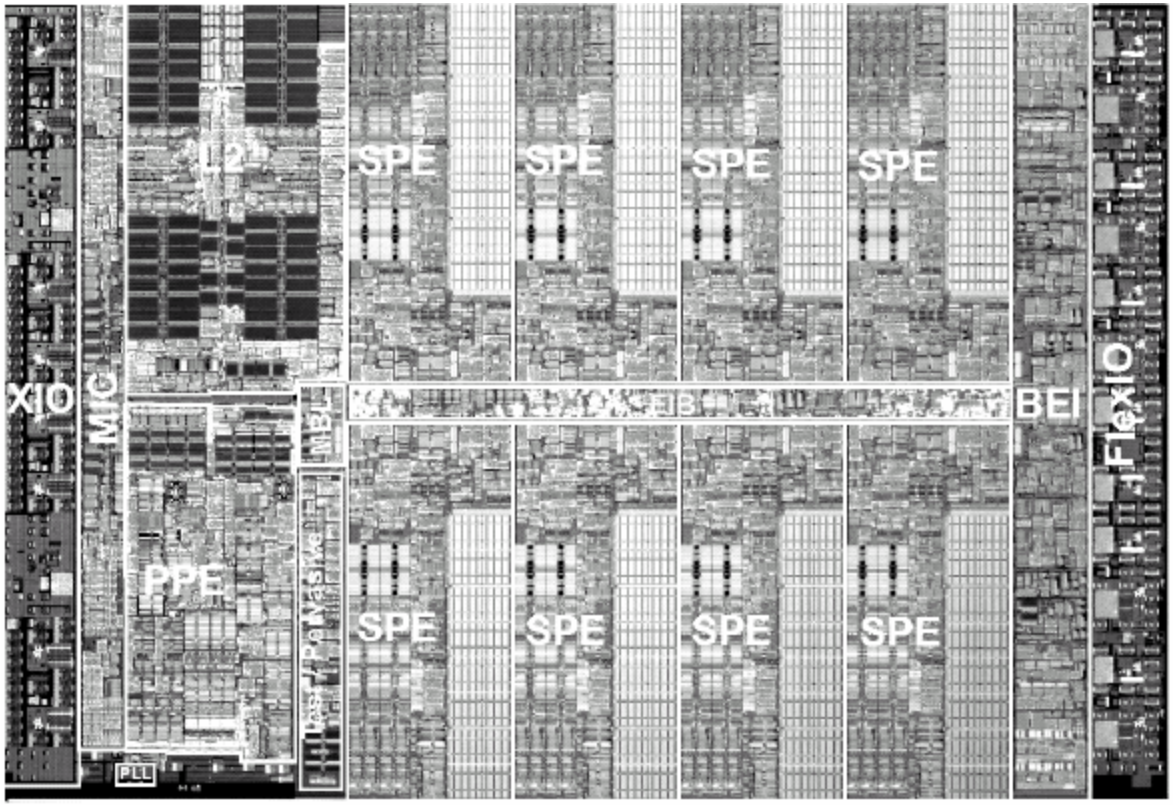
Application	Sustained GFLOPS	FP Ops / Mem Ref	LRF Refs	SRF Refs	Mem Refs
StreamFEM3D (Euler, quadratic)	31.6	17.1	153.0M (95.0%)	6.3M (3.9%)	1.8M (1.1%)
StreamFEM3D (MHD, constant)	39.2	13.8	186.5M (99.4%)	7.7M (0.4%)	2.8M (0.2%)
StreamMD (grid algorithm)	14.2*	12.1*	90.2M (97.5%)	1.6M (1.7%)	0.7M (0.8%)
GROMACS	38.8*	9.7*	108M (95.0%)	4.2M (2.9%)	1.5M (1.3%)
StreamFLO	12.9*	7.4*	234.3M (95.7%)	7.2M (2.9%)	3.4M (1.4%)

Simulated on a machine with 64GFLOPS peak performance and no fused MADD
* The low numbers are a result of many divide and square-root operations

Applications achieve high performance and make good use of the bandwidth hierarchy

Cell

- Cell is a stream processor
- “Local Store” in each SPE is equivalent to an SRF
- All of the software techniques we have developed can be applied to Cell



Conclusion: Explicit communication solves the hard problem: bandwidth

- Bandwidth is the critical resource (latency can be hidden)
  - Minimize demand
  - Keep it busy
- Explicit communication (storage) optimizes bandwidth
  - Producer-consumer locality reduces bandwidth demand
  - Latency well hidden – no misses – 1000s of outstanding references
  - Precise storage management
    - Fetch only needed data, No writes of dead data
- Stream compilation efficiently exploits explicit communication
  - Also enables simple, efficient ALU scheduling
- Merrimac establishes the feasiblity of this approach
  - Excellent simulated performance on wide range of scientific applications
- Cell is a stream processor w/ explicit communication
  - Stream compilation can be applied to cell-based machines.