

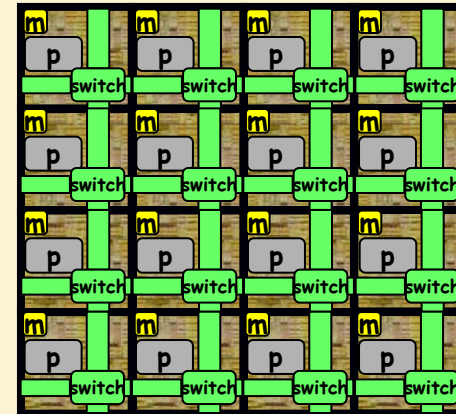
How will a Manycore Computing Environment Look

Anant Agarwal
MIT and Tiler

The 6 questions

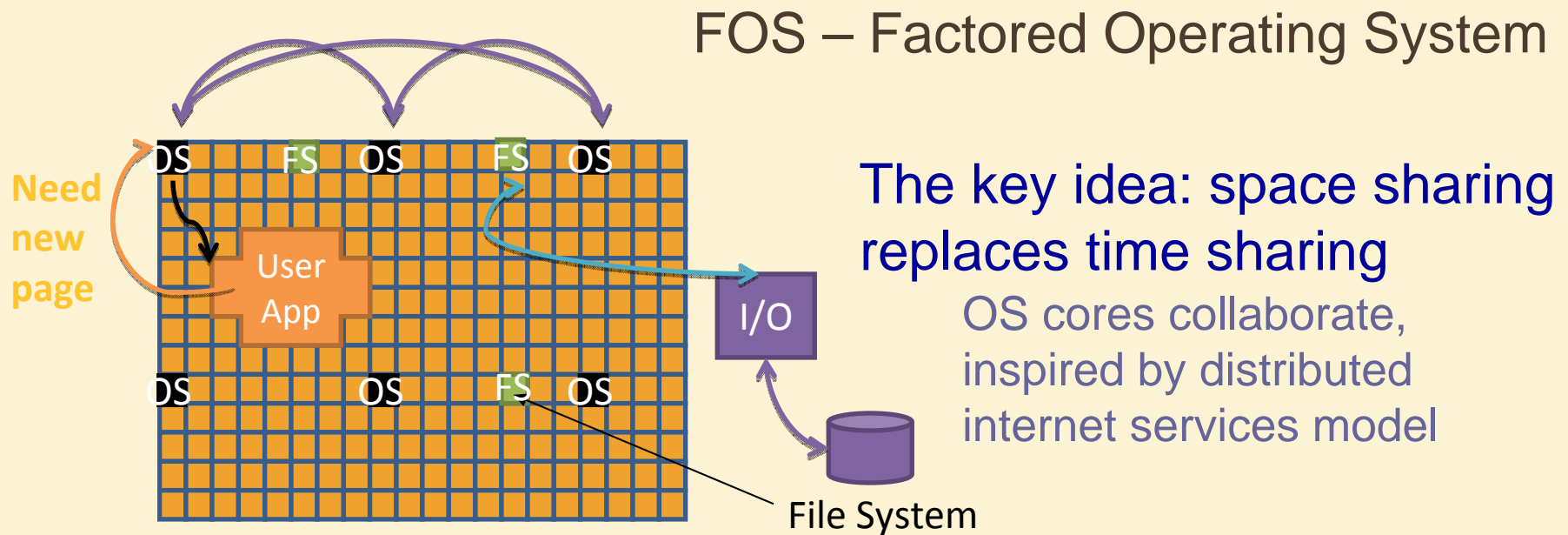
- I. HW Architecture
 - Homogeneous or heterogeneous? Memory architecture? Core composability?
- II. Operating system
 - Do we need an OS at all? Scheduling ?!
- III. Programming models
 - Threading? Pipelining? MP? Shared Memory? Else?
- IV. Programming languages
 - C/C++ rules? Domain specific languages?
- V. Legacy software
 - The best bet for scaling single-threaded application performance?
- VI. How will we get from here to the 'promised land of 1000s of core with easy programmability'?
 - The future is bright – but how is the road taking us there? What's in store the next 5 years?

1. Manycore Architecture



- ✘ Distributed everything – cores, memory, switching – for low power, simplicity, scalability
- ✘ Tiles are composable using Distributed ILP (as in MIT's Raw) – more later
- ✘ Cores are simpler, but self-respecting (can run an OS) -- **key for ease of programming** – use KILL rule to size cores
- ✘ Programmer visible cores will be homogeneous – otherwise too hard to program, design and build (note most multicores have been mostly homogeneous)

2. What OS

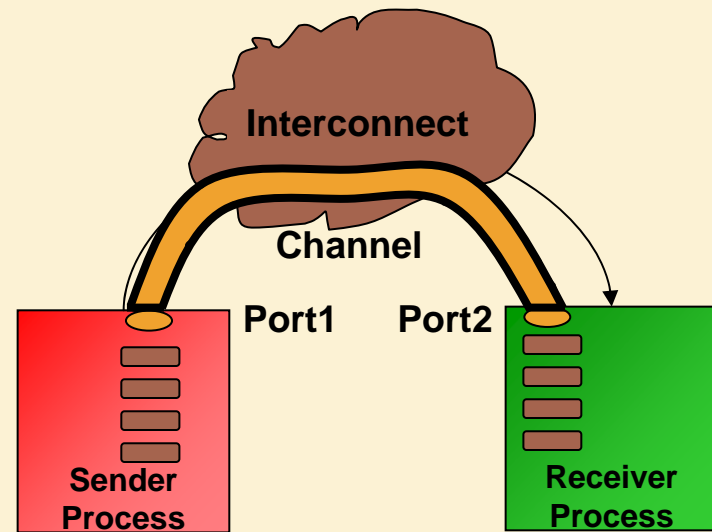


- Short term SMP Linux: User app and OS kernel thrash each other in a core's cache
 - User/OS time sharing is inefficient
- MIT's Project Angstrom: OS assumes abstracted space model. OS services bound to distinct cores, separate from user cores. OS service cores collaborate to achieve best resource management
 - User/OS space sharing is efficient

[Wentzlaff
Project Angstrom]

3. Programming Models and Languages

- ✘ Reality: In the short term, pthreads and shared memory
- ✘ Migrating to streaming, messaging, pipelining – these three are more composable approaches
- ✘ Streaming:
 - + Familiar, like sockets
 - + Basis of networking and internet software
 - + Easy to support both pipelined and data parallel models
 - + Modular & scalable
 - + Spatial relationships can be captured
 - + E.g., Multicore association’s MCAPL is a step in this direction



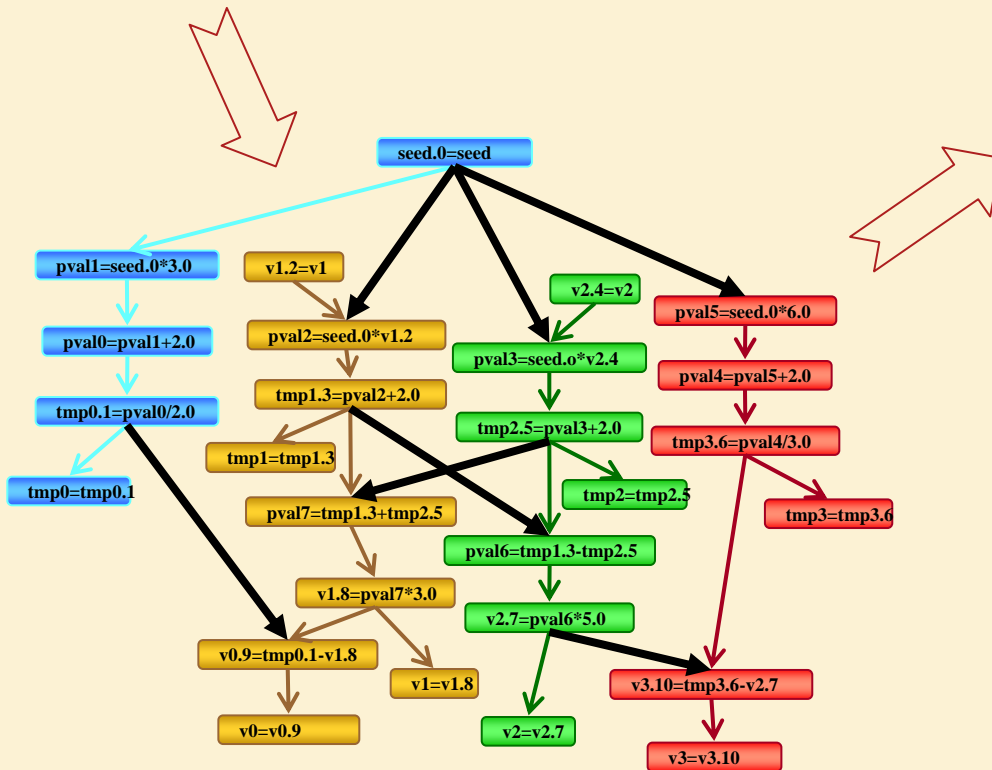
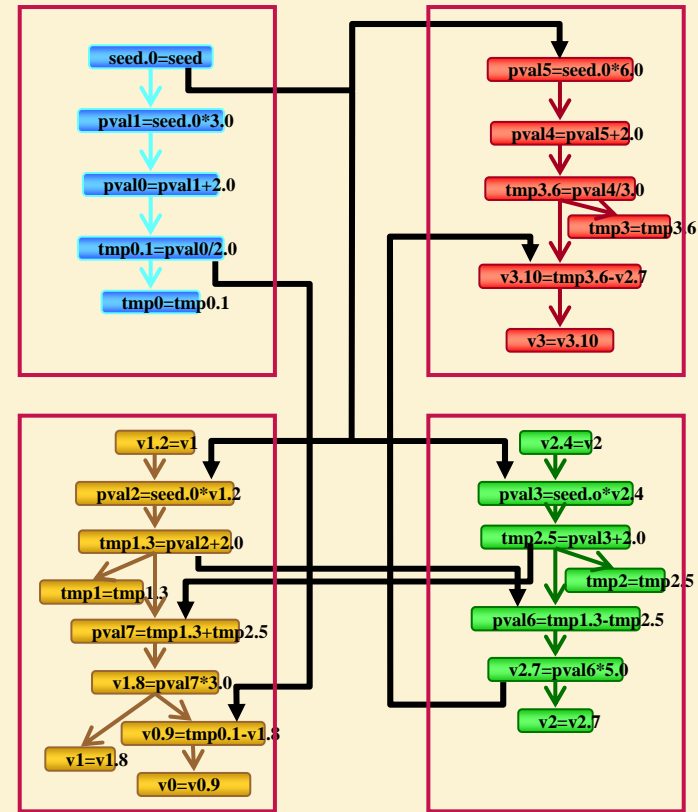
Streaming is Familiar – Like Sockets

5. Scaling Single Thread Performance: Distributed ILP (DILP) Approach

```

tmp0 = (seed*3+2)/2
tmp1 = seed*v1+2
tmp2 = seed*v2 + 2
tmp3 = (seed*6+2)/3
v2 = (tmp1 - tmp3)*5
v1 = (tmp1 + tmp2)*3
v0 = tmp0 - v1
v3 = tmp3 - v2
    
```

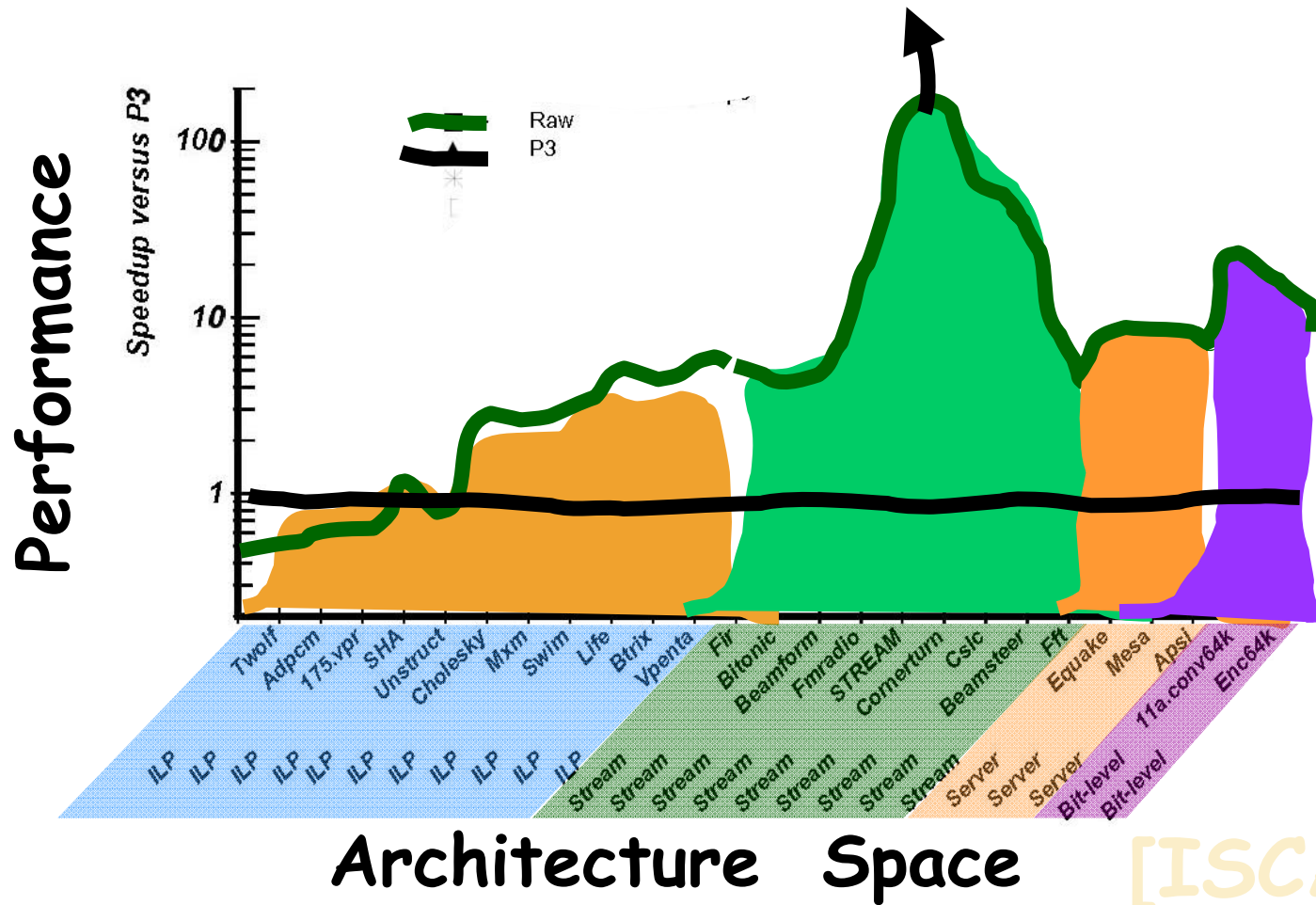
Combining Multiple Cores to Behave Like One More Powerful Core



Raw [ISCA04]

DILP Performance Results

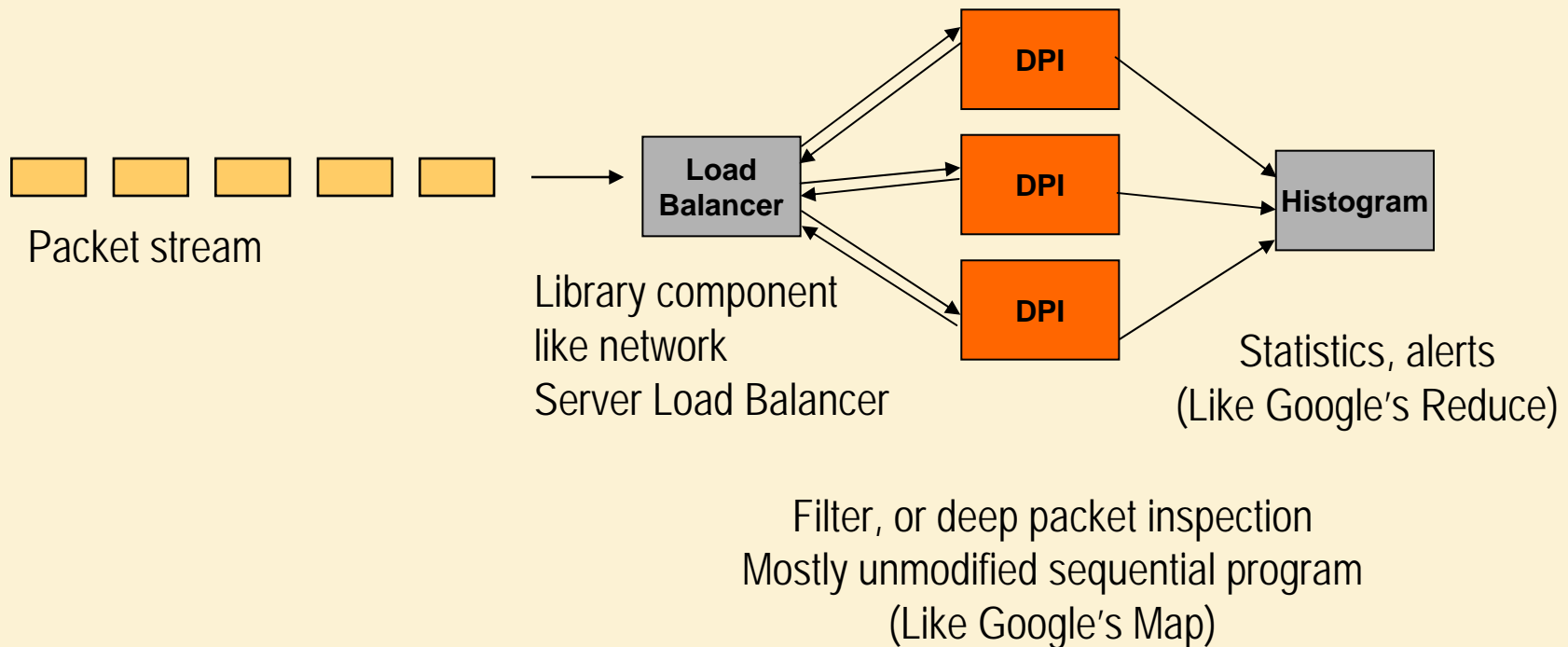
- ~10x parallelism
- ~ 4x ld/store elim
- ~ 4x stream mem bw



[ISCA'04]

4, 6. Legacy SW, Incremental Path to the Future, Domain Specific Languages

Like Google's Map-Reduce



Mostly sequential programming!