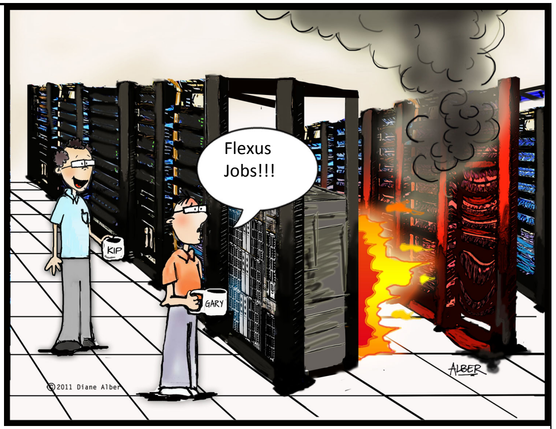
ACA

Servers

Fall 2016



Pejman Lotfi-Kamran

Adapted from slides originally developed by Profs. Hill, Hoe, Falsafi
and Wenisch of CMU, EPFL, Michigan, Wisconsin

Where Are We?

Fr	Sa	Su	Mo	Tu
9/26	27-Shahrivar	9/28	29-Shahrivar	9/30
10/2	3-Mehr	10/4	5-Mehr	10/6
10/9	10-Mehr	10/11	12-Mehr	###
10/16	17-Mehr	10/18	19-Mehr	###
10/23	24-Mehr	10/25	26-Mehr	###
10/31	1-Aban	11/2	3-Aban	11/4
11/7	8-Aban	11/9	10-Aban	###
11/14	15-Aban	11/17	17-Aban	###
11/21	22-Aban	11/23	24-Aban	###
11/28	29-Aban	11/30	1-Azar	12/2
12/5	6-Azar	12/7	8-Azar	12/9
12/12	13-Azar	12/1	15-Azar	###
12/19	20-Azar		22-Azar	###
12/26	27-Azar	12/2	29-Azar	###
1/3	4-Dey	1/5	6-Dey	1/7

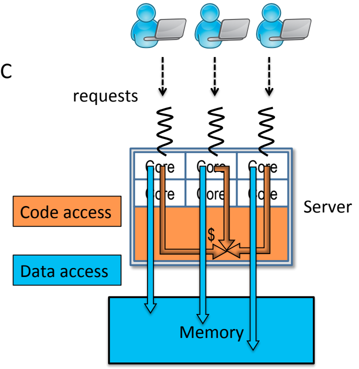
- This Lecture
  - Server
- Next Lecture:
  - Data Center

What do Scale-Out Apps Need? [ISCA 2012] [IEEE Micro 2012]

Cores share instructions
- Large code footprint fits in LLC

Data is in memory
- Data footprint dwarfs LLC

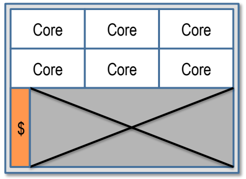
Cores don't communicate
- Independent requests
- Mostly Read-only accesses



Poor match to existing server processors

Conventional Processors: Major Mismatch

- ✘ Few fat cores
  - Limited on-chip parallelism
- ✘ Large LLC
  - Dwarfed by vast datasets
  - Slow to access
  - Takes up much die area

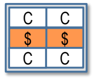


e.g., *Intel Westmere*

Large cores & LLC wasteful on scale-out apps

Small-chip Processors: Not Enough

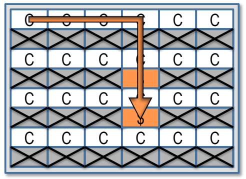
- ✘ Few lean cores
  - – Low parallelism
- ✘ Modestly-sized cache
  - – Fast to access, but…
  - – takes up large fraction of die area



e.g., *Calxeda*

Emerging Tiled Processors: Not Optimal

- ✓ Many lean cores
  - – High parallelism
- ✘ Large distributed cache
  - – Dwarfed by vast datasets
  - – Slow to access
  - – Takes up much die area
- ✘ Tiled organization
  - – Distance hurts performance



e.g., *Tilera Tile64*

Large LLC & distance prevent scalability

Outline

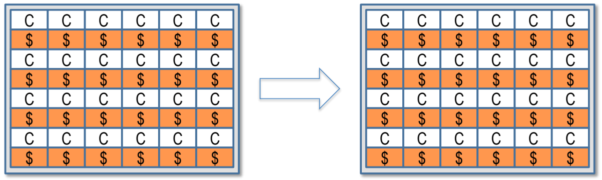
- Why not today's processors?
- What do scale-out workloads need?
- Scale-Out Processors
  - – Overview
  - – Design considerations

Roadmap for Scale-Out Processor

1. Eliminate wasteful capacity in the LLC
   - – Shrink the cache, add cores in freed space
2. Reduce delay to LLC
   - – Partition the die into independent multi-core *pods*
3. Enable technology scalability
   - – Do not interconnect the pods

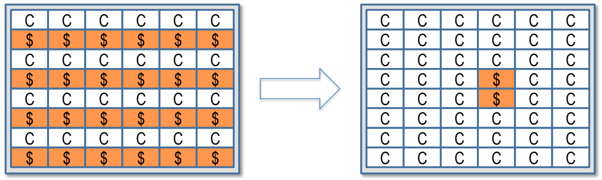
Step 1: Less cache, more cores

- ✓ Small LLC
  - Capture instructions
  - More area for the cores



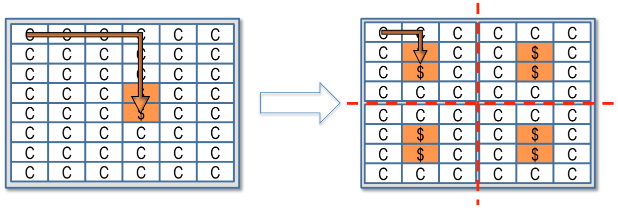
Step 1: Less cache, more cores

- ✓ Small LLC
  - Capture instructions
  - More area for the cores
- ✓ More cores for higher throughput



Step 2: Form “pods” to reduce distance

- ✓ Small LLC
- ✓ More cores for higher throughput
- ✘ Fast path to LLC for instructions



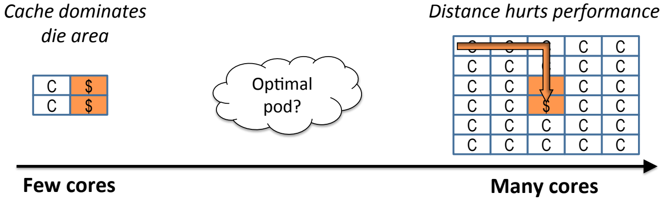
How to choose the optimal pod?

Sizing Pods: Not So Simple

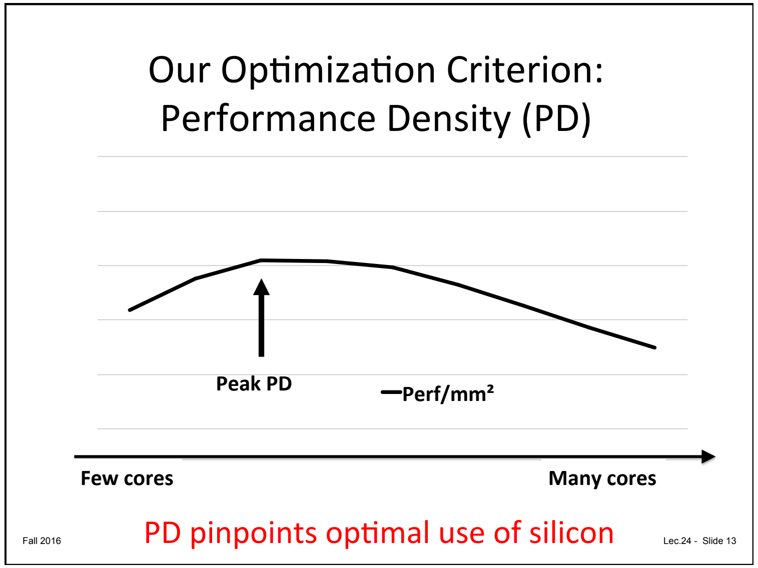
Too few cores ➔ limited parallelism

Too many cores ➔ long distance to LLC

- Slower instruction fetch

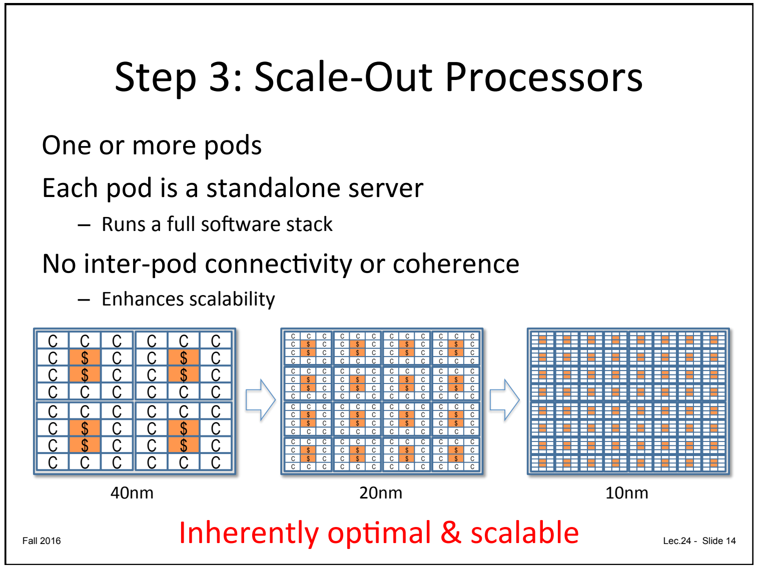


How do we characterize optimality?



Our Optimization Criterion: Performance Density (PD)

PD pinpoints optimal use of silicon



Step 3: Scale-Out Processors

One or more pods

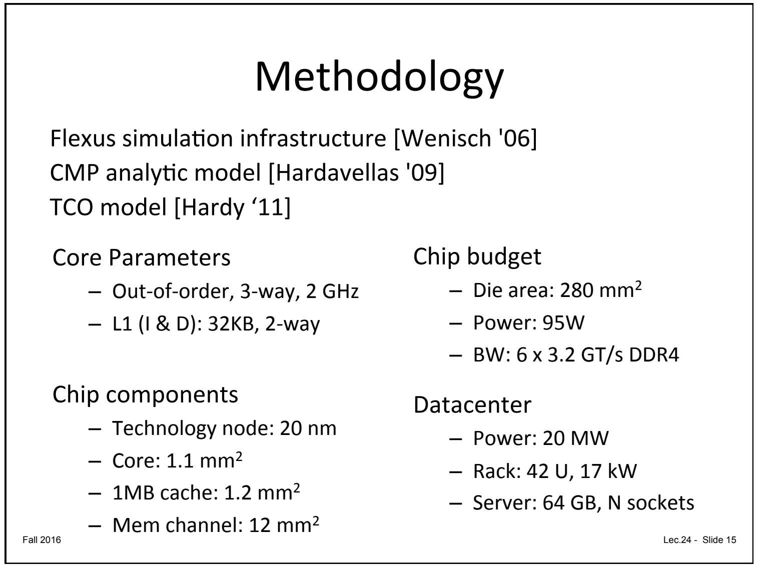
Each pod is a standalone server

- Runs a full software stack

No inter-pod connectivity or coherence

- Enhances scalability

Inherently optimal & scalable



Methodology

Flexus simulation infrastructure [Wenisch '06]
CMP analytic model [Hardavellas '09]
TCO model [Hardy ‘11]

Core Parameters

- Out-of-order, 3-way, 2 GHz
- L1 (I & D): 32KB, 2-way

Chip components

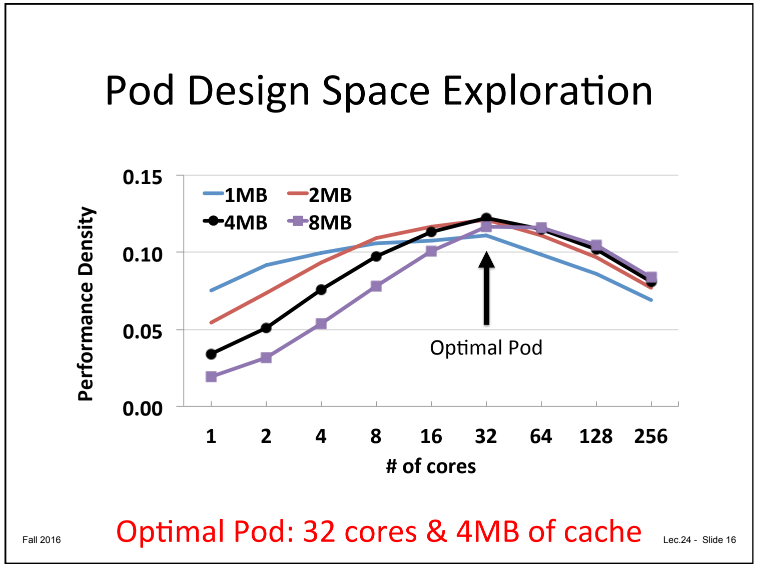
- Technology node: 20 nm
- Core: 1.1 mm^2
- 1MB cache: 1.2 mm^2
- Mem channel: 12 mm^2

Chip budget

- Die area: 280 mm^2
- Power: 95W
- BW: 6 x 3.2 GT/s DDR4

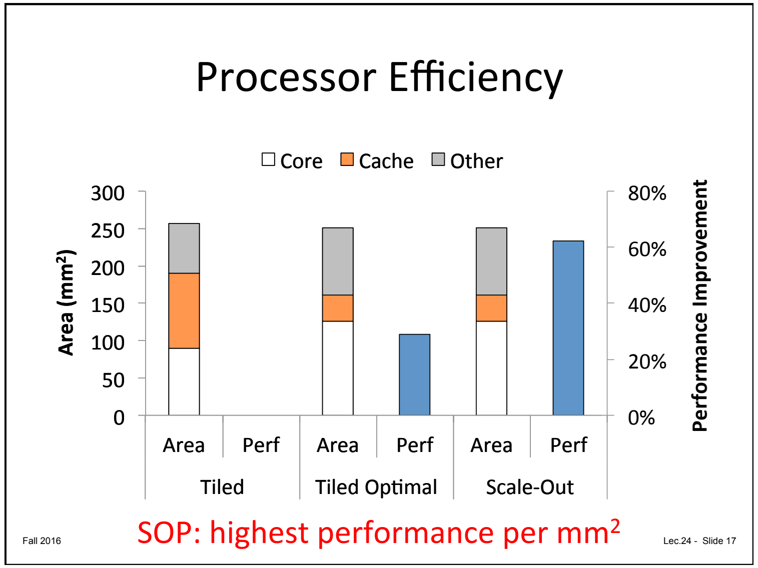
Datacenter

- Power: 20 MW
- Rack: 42 U, 17 kW
- Server: 64 GB, N sockets



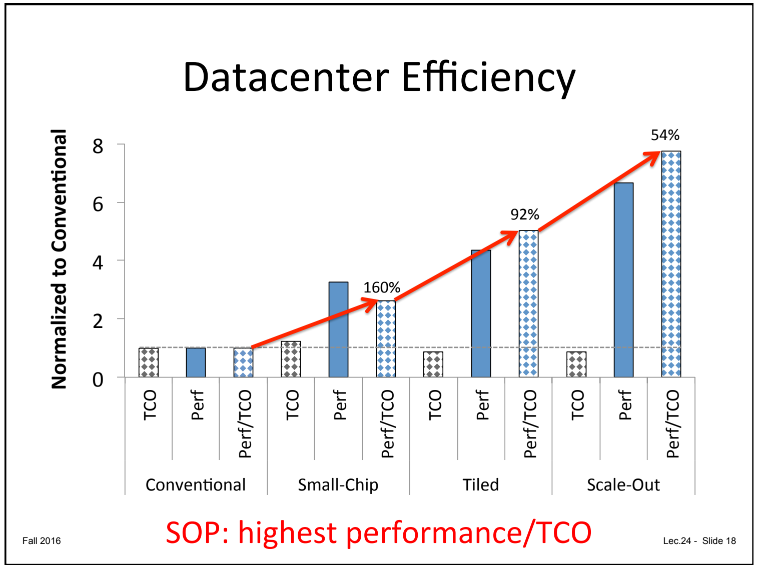
Pod Design Space Exploration

Optimal Pod: 32 cores & 4MB of cache



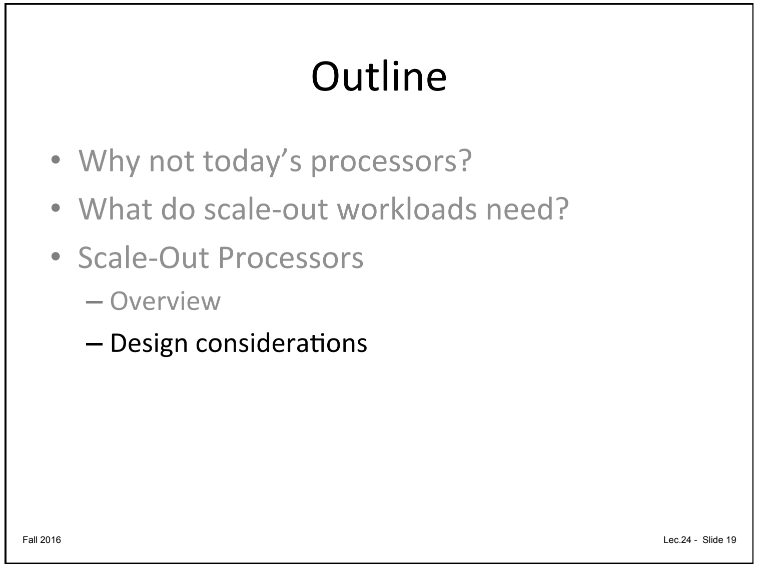
Processor Efficiency

SOP: highest performance per mm^2



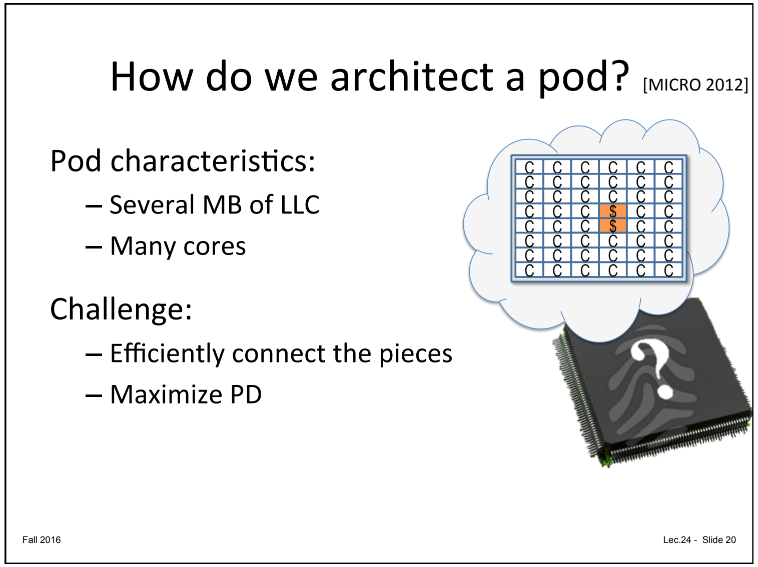
Datacenter Efficiency

SOP: highest performance/TCO



Outline

- Why not today's processors?
- What do scale-out workloads need?
- Scale-Out Processors
  – Overview
  – Design considerations



How do we architect a pod? [MICRO 2012]

Pod characteristics:

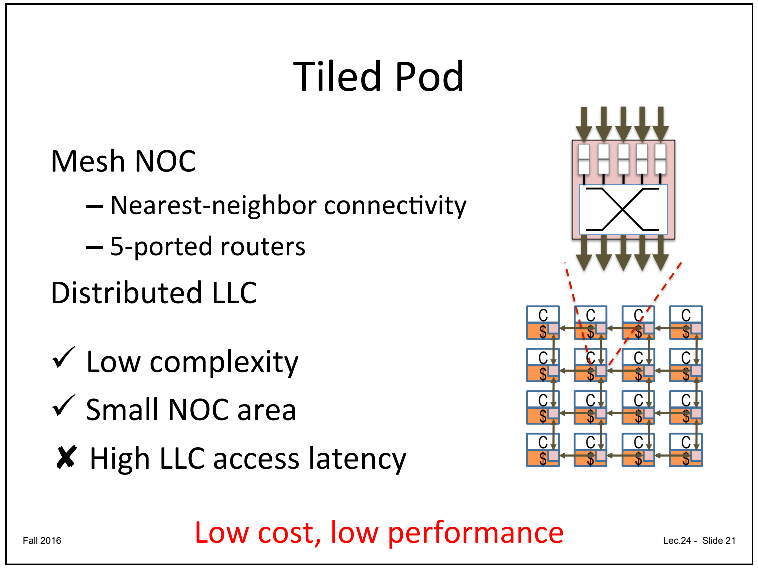
– Several MB of LLC

– Many cores

Challenge:

– Efficiently connect the pieces

– Maximize PD



Tiled Pod

Mesh NOC

- Nearest-neighbor connectivity
- 5-ported routers

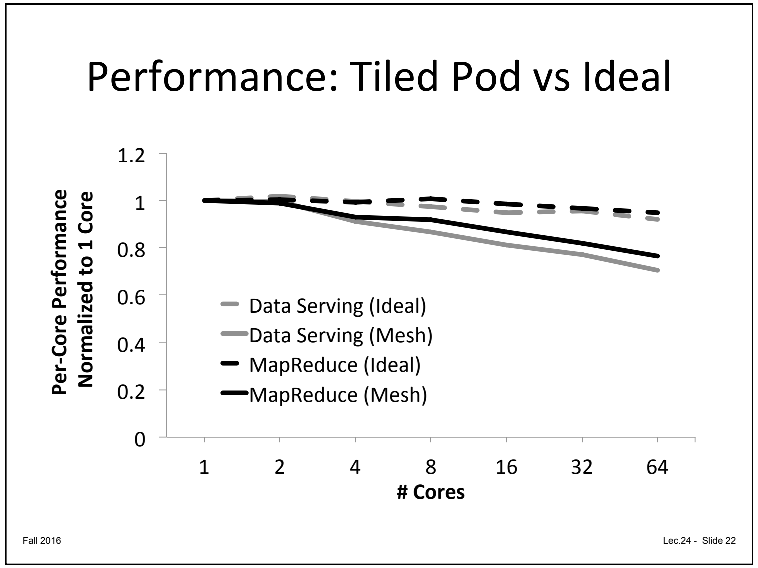
Distributed LLC

✓ Low complexity

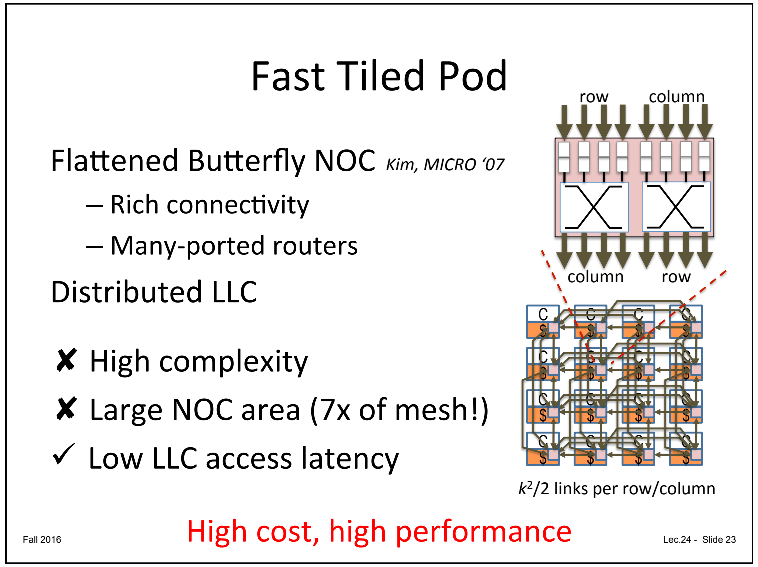
✓ Small NOC area

✘ High LLC access latency

Low cost, low performance



Performance: Tiled Pod vs Ideal



Fast Tiled Pod

Flattened Butterfly NOC *Kim, MICRO '07*

- Rich connectivity
- Many-ported routers

Distributed LLC

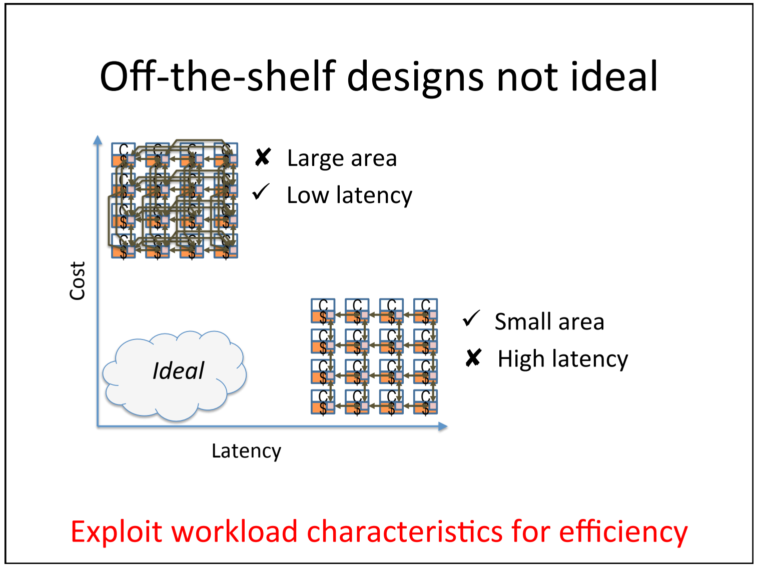
✘ High complexity

✘ Large NOC area (7x of mesh!)

✓ Low LLC access latency

$k^2/2$ links per row/column

High cost, high performance



Off-the-shelf designs not ideal

✘ Large area

✓ Low latency

✓ Small area

✘ High latency

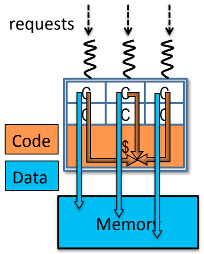
Exploit workload characteristics for efficiency

How do threads access data?

Where is the code and data?

- Code: in LLC
- Data: in memory

➢Nothing useful in remote L1s!



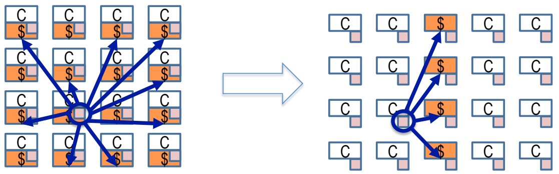
Data access pattern: *bilateral*

- L1 ↔ LLC
- L1 ↮ L1

What is the actual traffic pattern?

Distributed LLC: all-to-all traffic

- Mismatch with underlying bilateral access pattern
- Requires expensive connectivity for low latency



Turn all-to-all traffic into bilateral to reduce cost!

Roadmap for an "Ideal" Pod

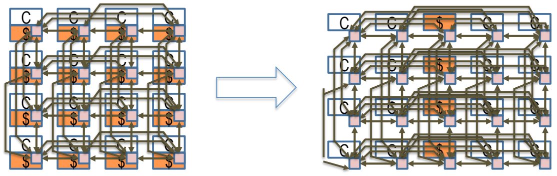
Starting point: fast tiled pod

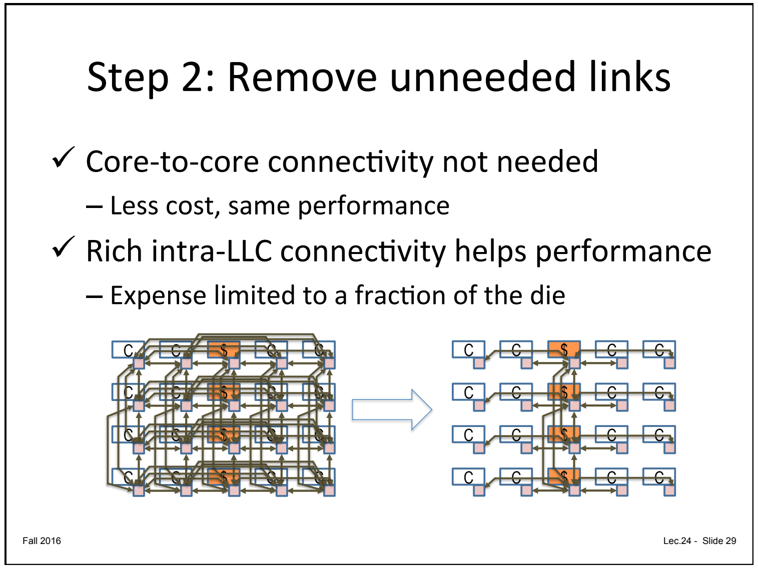
1. LLC in the center
   - Decouple cores and LLC banks
2. Remove links from flattened butterfly
   - Leverage the *bilateral* traffic pattern to lower cost
3. Share and specialize core-to-LLC interconnect

Step 1: LLC in the Center

✓ Decouple core and LLC tiles

- Natural fit for the bilateral traffic pattern



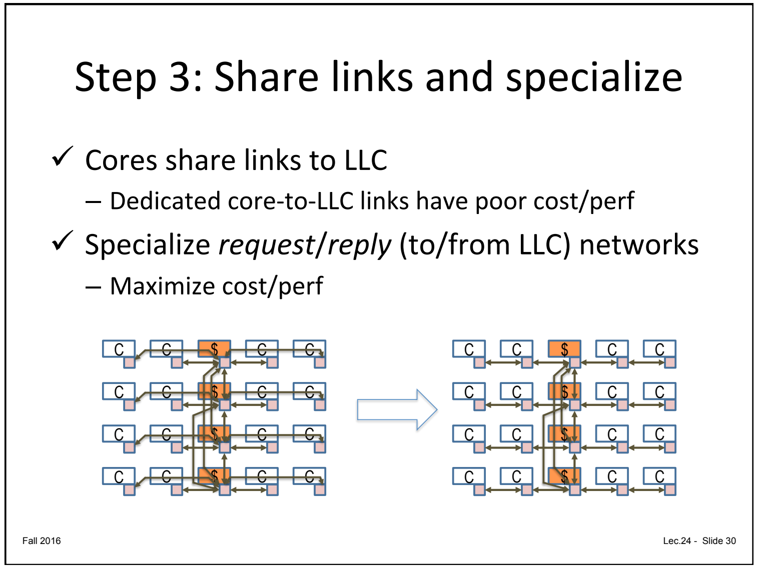


Step 2: Remove unneeded links

- ✓ Core-to-core connectivity not needed
  - – Less cost, same performance
- ✓ Rich intra-LLC connectivity helps performance
  - – Expense limited to a fraction of the die

Fall 2016

Lec.24 - Slide 29

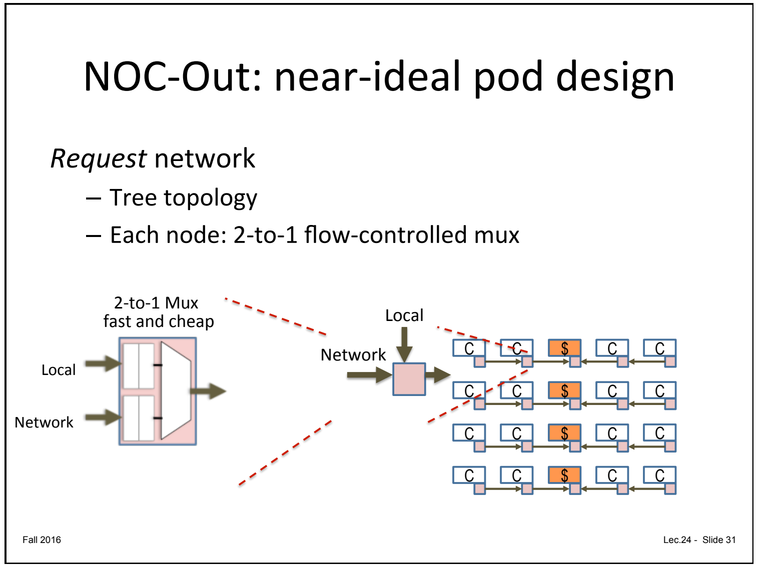


Step 3: Share links and specialize

- ✓ Cores share links to LLC
  - – Dedicated core-to-LLC links have poor cost/perf
- ✓ Specialize *request/reply* (to/from LLC) networks
  - – Maximize cost/perf

Fall 2016

Lec.24 - Slide 30



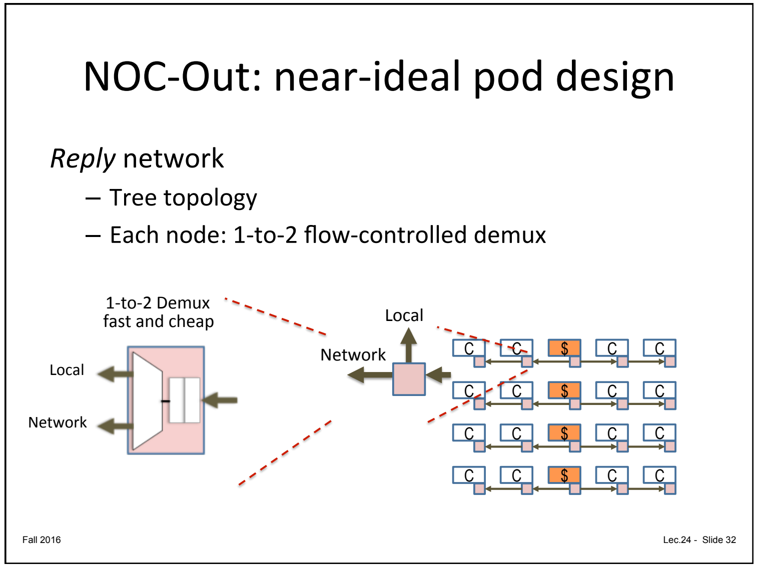
NOC-Out: near-ideal pod design

Request network

- – Tree topology
- – Each node: 2-to-1 flow-controlled mux

Fall 2016

Lec.24 - Slide 31



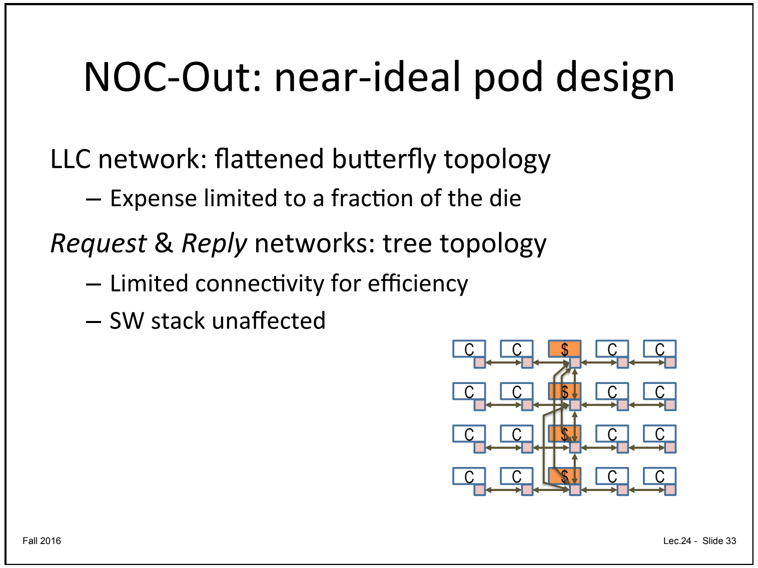
NOC-Out: near-ideal pod design

Reply network

- – Tree topology
- – Each node: 1-to-2 flow-controlled demux

Fall 2016

Lec.24 - Slide 32



NOC-Out: near-ideal pod design

LLC network: flattened butterfly topology

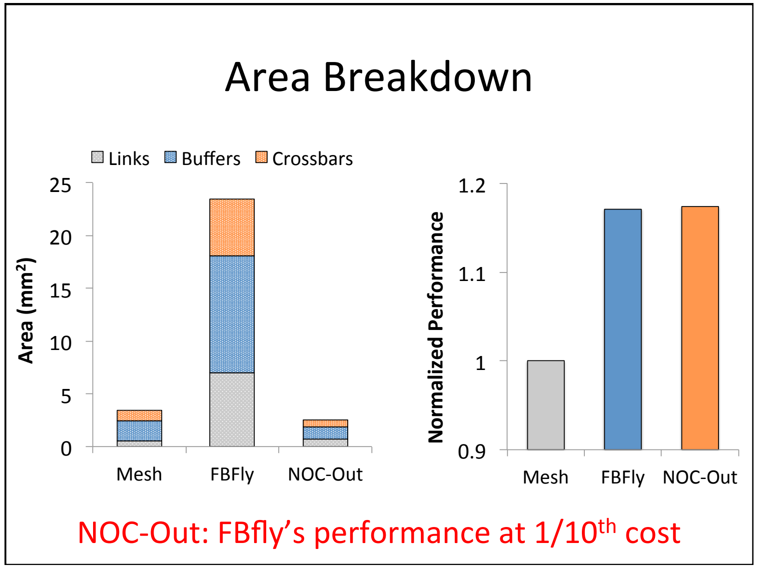
- Expense limited to a fraction of the die

Request & *Reply* networks: tree topology

- Limited connectivity for efficiency
- SW stack unaffected

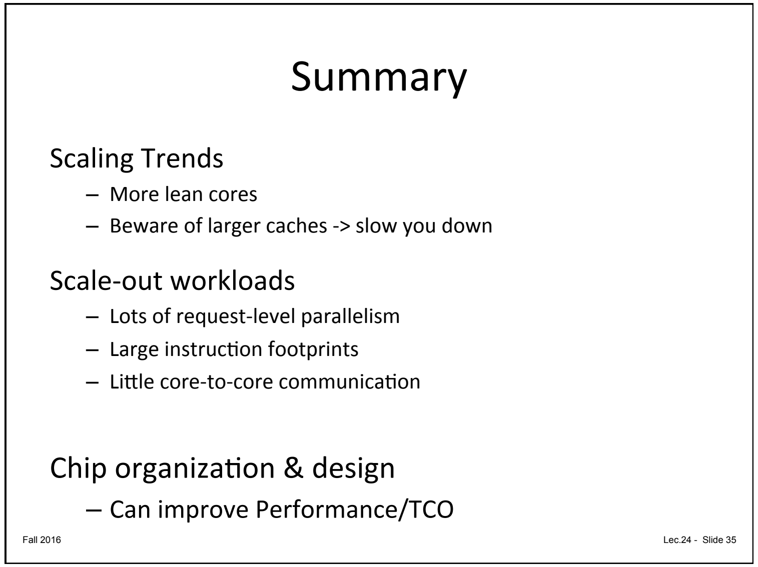
Fall 2016

Lec.24 - Slide 33



Area Breakdown

NOC-Out: FBfly's performance at 1/10th cost



Summary

Scaling Trends

- More lean cores
- Beware of larger caches -> slow you down

Scale-out workloads

- Lots of request-level parallelism
- Large instruction footprints
- Little core-to-core communication

Chip organization & design

- Can improve Performance/TCO

Fall 2016

Lec.24 - Slide 35