COMP 322: Fundamentals of Parallel Programming

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Lecture 22: Task Affinity with Places

Vivek Sarkar
Department of Computer Science
Rice University
vsarkar@rice.edu

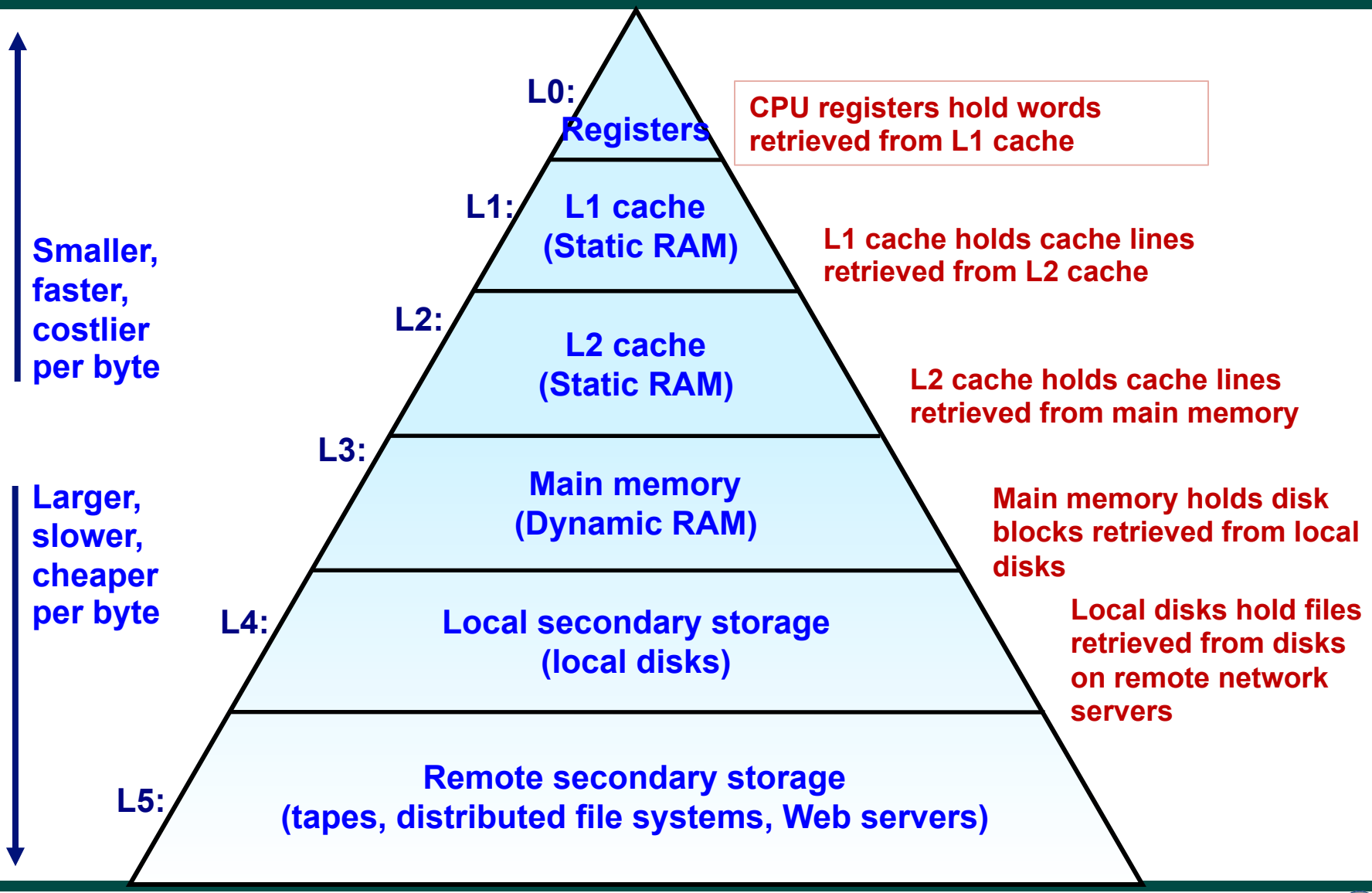
Announcements

- Homework 5 (written assignment) due by 5pm on Friday, March 18th
- Homework 6 (HJ programming assignment) will be assigned on March 18th
- Homework 7 (Concurrent Java programming assignment) will be assigned on April 1st (really!)

Acknowledgments for Today's Lecture

- Lecture 22 handout
- Randal E. Bryant & David R. O'Hallaron. Computer Systems: A Programmer's Perspective, Second Edition. Prentice Hall, 2010.
  - Selected slides extracted from http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/schedule.html

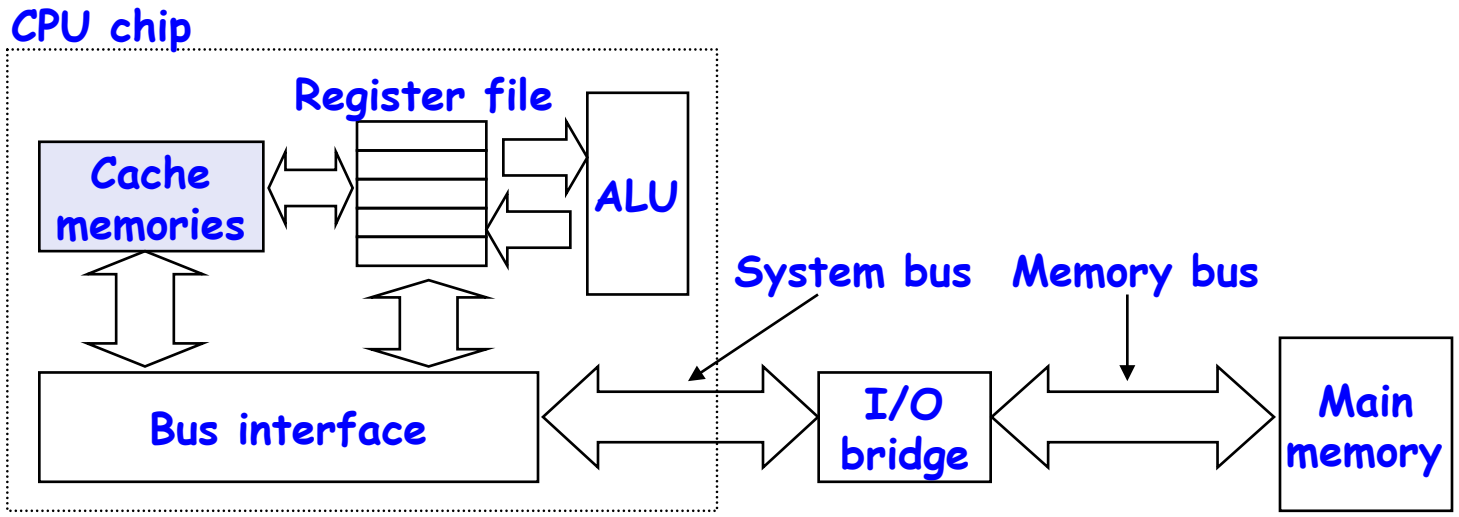
An example Memory Hierarchy --- what is the cost of a Memory Access?



 Source: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/09-memory-hierarchy.pptx

Cache Memories

- Cache memories are small, fast SRAM-based memories managed automatically in hardware.
  - Hold frequently accessed blocks of main memory
- CPU looks first for data in caches (e.g., L1, L2, and L3), then in main memory.
- Typical system structure:



Source: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/09-memory-hierarchy.pptx

Storage Trends

SRAM

Metric	1980	1985	1990	1995	2000	2005	2010	*2010:1980*
$/MB	19,200	2,900	320	256	100	75	60	*320*
access (ns)	300	150	35	15	3	2	1.5	*200*

DRAM

Metric	1980	1985	1990	1995	2000	2005	2010	*2010:1980*
$/MB	8,000	880	100	30	1	0.1	0.06	*130,000*
access (ns)	375	200	100	70	60	50	40	*9*
typical size (MB)	0.064	0.256	4	16	64	2,000	8,000	*125,000*

Disk

Metric	1980	1985	1990	1995	2000	2005	2010	*2010:1980*
$/MB	500	100	8	0.30	0.01	0.005	0.0003	*1,600,000*
access (ms)	87	75	28	10	8	*4*	*3*	*29*
typical size (MB)	1	10	160	1,000	20,000	160,000	1,500,000	*1,500,000*

Source: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/09-memory-hierarchy.pptx

Table 1: Examples of Caching in the Hierarchy

Hierarchy Level	What is cached?	Where is it cached?	Latency (cycles)	Managed by
Registers	4-32 bytes (words)	CPU core	0	Compiler
TLB	Address translations	On-chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	$O(10^0)$	Hardware
L2 cache	64-bytes block	On/Off-Chip L2	$O(10^1)$	Hardware
Virtual Memory	4 KB page	Main memory	$O(10^2)$	Hardware & OS
Buffer cache	Parts of files	Main memory	$O(10^2)$	OS
Disk cache	Disk sectors	Disk controller	$O(10^5)$	Disk firmware
Network buffer cache	Parts of files	Local disk	$O(10^7)$	AFS/NFS client
Browser cache	Web pages	Local disk	$O(10^7)$	Web browser
Web cache	Web pages	Remote server disks	$O(10^9)$	Web proxy server

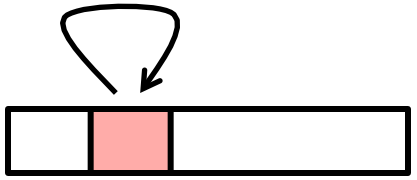
Ultimate goal: create a large pool of storage with average cost per byte that approaches that of the cheap storage near the bottom of the hierarchy, and average latency that approaches that of fast storage near the top of the hierarchy.

Source: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/09-memory-hierarchy.pptx

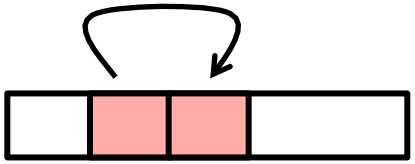
Locality

- **Principle of Locality:** Programs tend to use data and instructions with addresses near or equal to those they have used recently

- **Temporal locality:**
  - Recently referenced items are likely to be referenced again in the near future



- **Spatial locality:**
  - Items with nearby addresses tend to be referenced close together in time



Source: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/09-memory-hierarchy.pptx

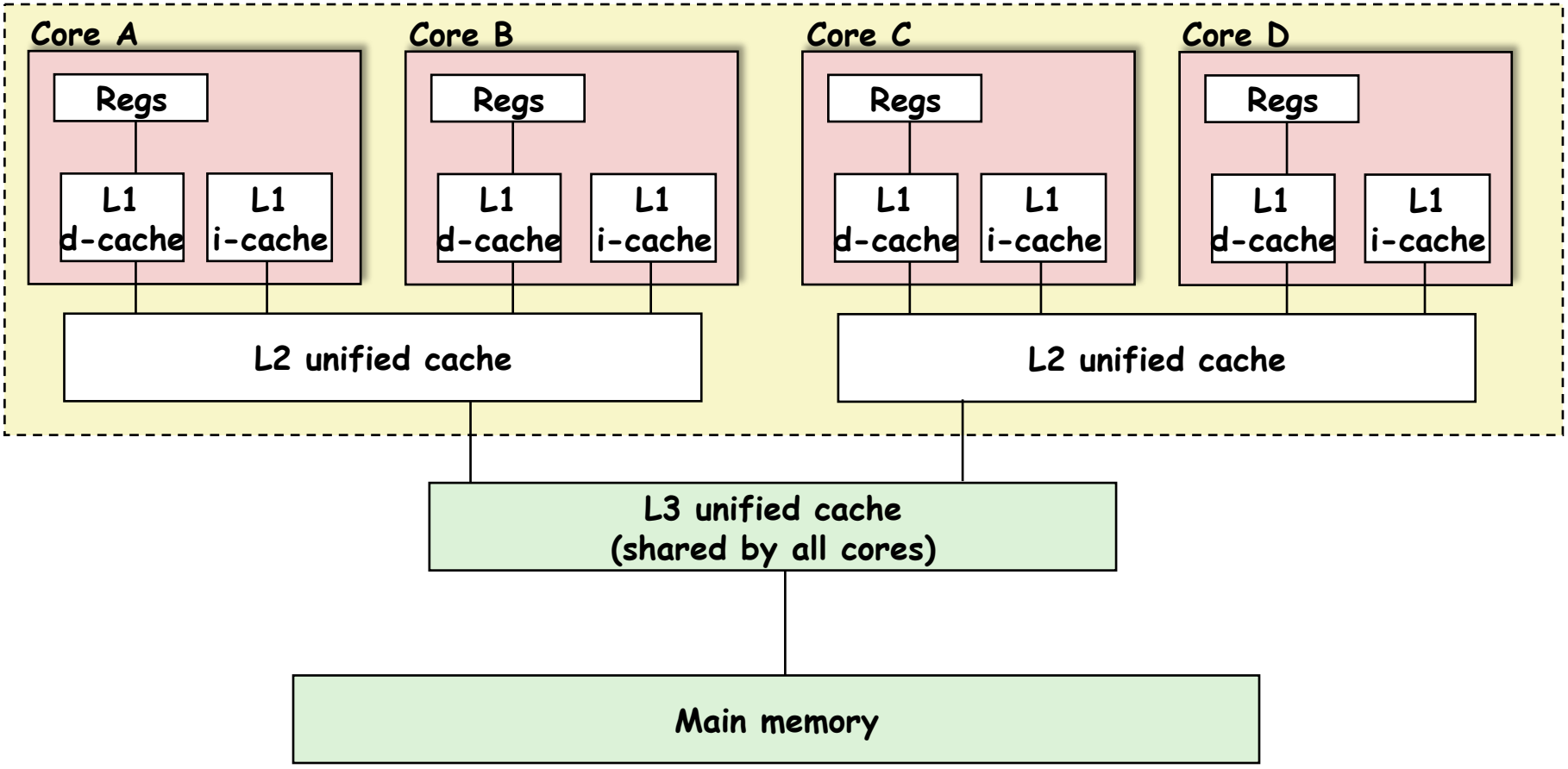
Locality Example

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- Data references
  - Reference array elements in succession (stride-1 reference pattern). — **Spatial locality**
  - Reference variable `sum` each iteration. — **Temporal locality**
- Instruction references
  - Reference instructions in sequence. — **Spatial locality**
  - Cycle through loop repeatedly. — **Temporal locality**

Source: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/09-memory-hierarchy.pptx

Memory Hierarchy in a Multicore Processor



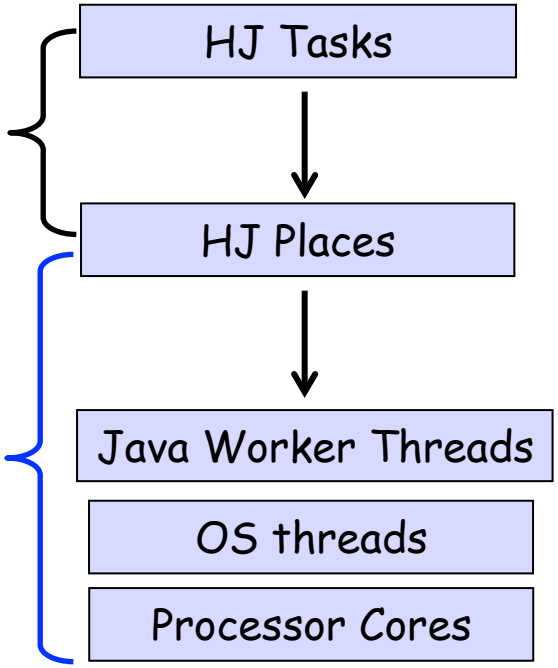
- Memory hierarchy for a single Intel Xeon Quad-core E5440 HarperTown processor chip
  - A SUG@R node contains two such chips

Programmer Control of Task Assignment to Processors

- The parallel programming constructs that we've studied thus far for task creation result in tasks that are assigned to processors *dynamically* by the HJ runtime system
  - Programmer does not have to worry about task assignment details
- Sometimes, programmer control of task assignment can lead to significant performance advantages due to improved locality
- Motivation for HJ "places"
  - Provide the programmer a mechanism to map each task to a set of core when the task is created

Places in HJ

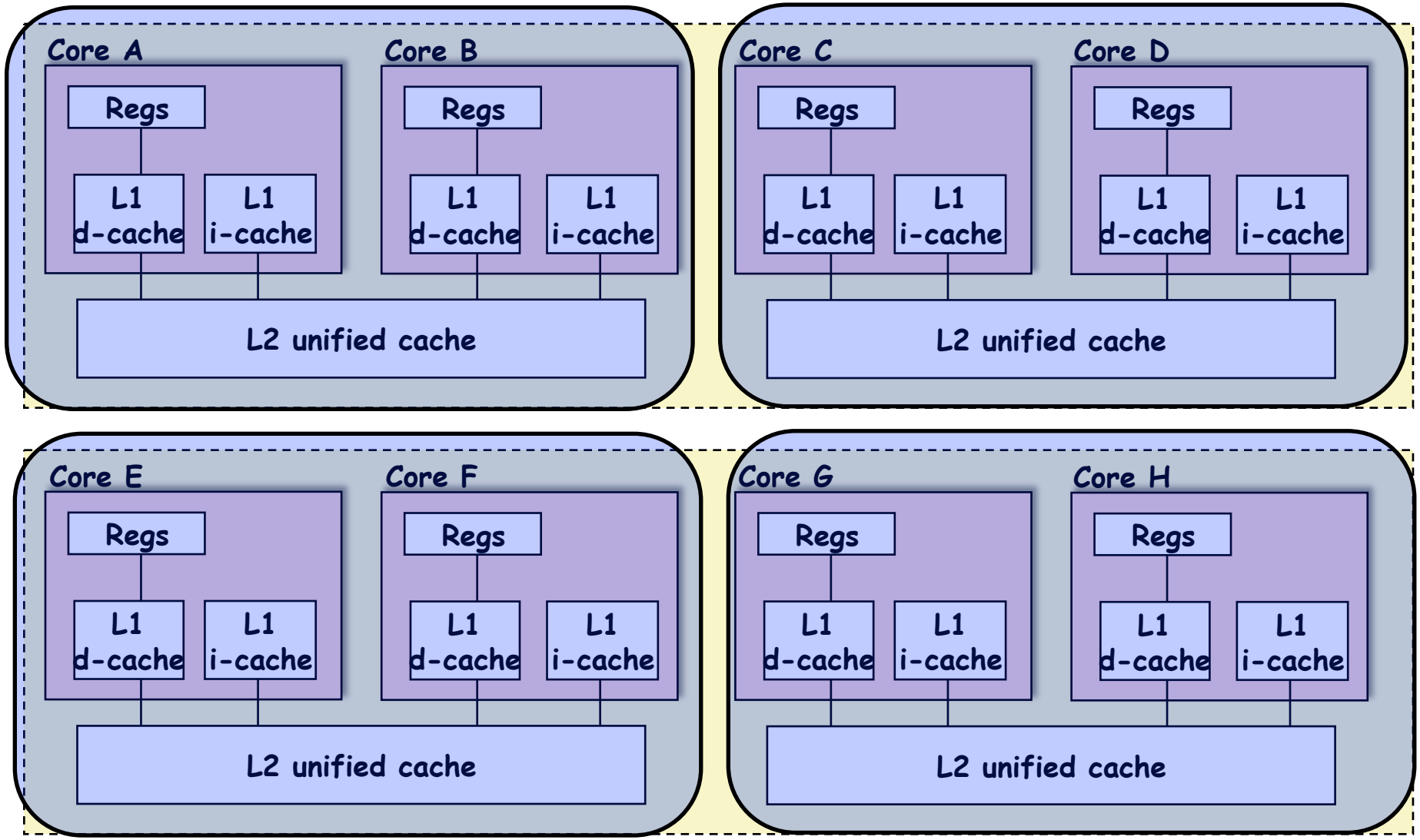
HJ programmer defines mapping from HJ tasks to set of places



HJ runtime defines mapping from places to one or more worker Java threads per place

The option "-places p:w" when executing an HJ program can be used to specify
p, the number of places
w, the number of worker threads per place

Example of –places 4:2 option on a SUG@R node (4 places w/ 2 workers per place)



Places in HJ

here = place at which current task is executing

place.MAX_PLACES = total number of places (runtime constant)

Specified by value of p in runtime option, `-places p:w`

place.factory.place(i) = place corresponding to index i

<place-expr>.toString() returns a string of the form "place(id=0)"

<place-expr>.id returns the id of the place as an int

async at(P) S

- Creates new task to execute statement S at place P
- **async S** is equivalent to **async at(here) S**

Note that **here** in a child task for an async/future computation will refer to the place P at which the child task is executing, not the place where the parent task is executing

Listing 1: Example HJ program with places

```
1  class T1 {
2    final place affinity;
3    . . .
4    // T1's constructor sets affinity to place where instance was created
5    T1() { affinity = here; ... }
6    . . .
7  }
8  . . .
9  finish { // Inter-place parallelism
10   System.out.println("Parent place = ", here); // Parent task s place
11   for (T1 a = . . .) {
12     async at (a.affinity) { // Execute async at place with affinity to a
13       a.foo();
14       System.out.println("Child place = ", here); // Child task's place
15     } // async
16   } // for
17 } // finish
18 . . .
```