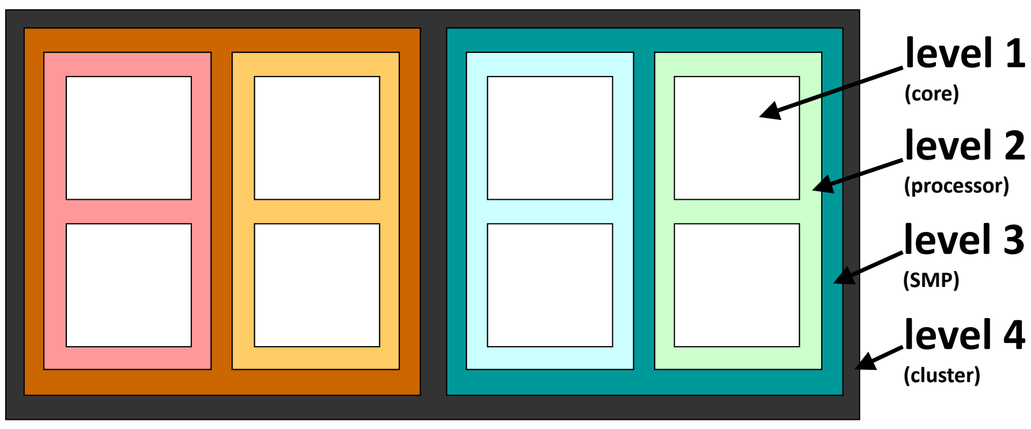
Titanium Hierarchical Extensions to the PGAS/SPMD Programming Model

PGAS

Overview

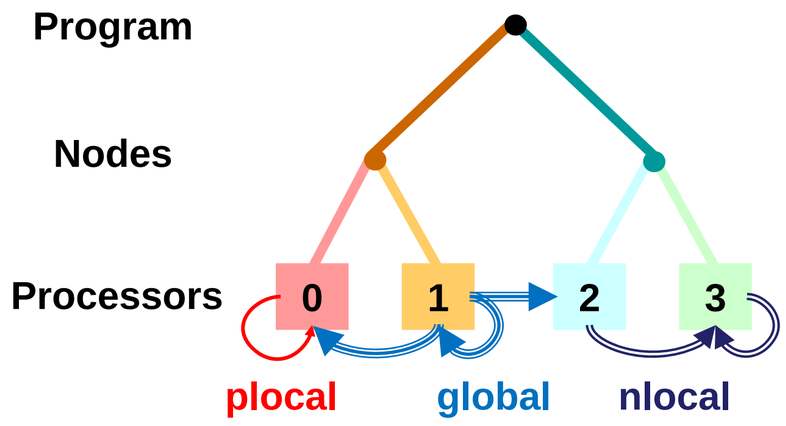
- **Parallel machines have hierarchical structure**
  - **Communication costs depend on location of sender, receiver**



- **Memory hierarchy encapsulates communication costs in memory model**
- **Control hierarchy allows pro-grammer to arrange computation to minimize communication costs**

Memory Hierarchy

- **Type system qualifies pointers with their maximum *reach***
  - **Global pointers can reference data anywhere, processor-local pointers can only reference data on same processor**



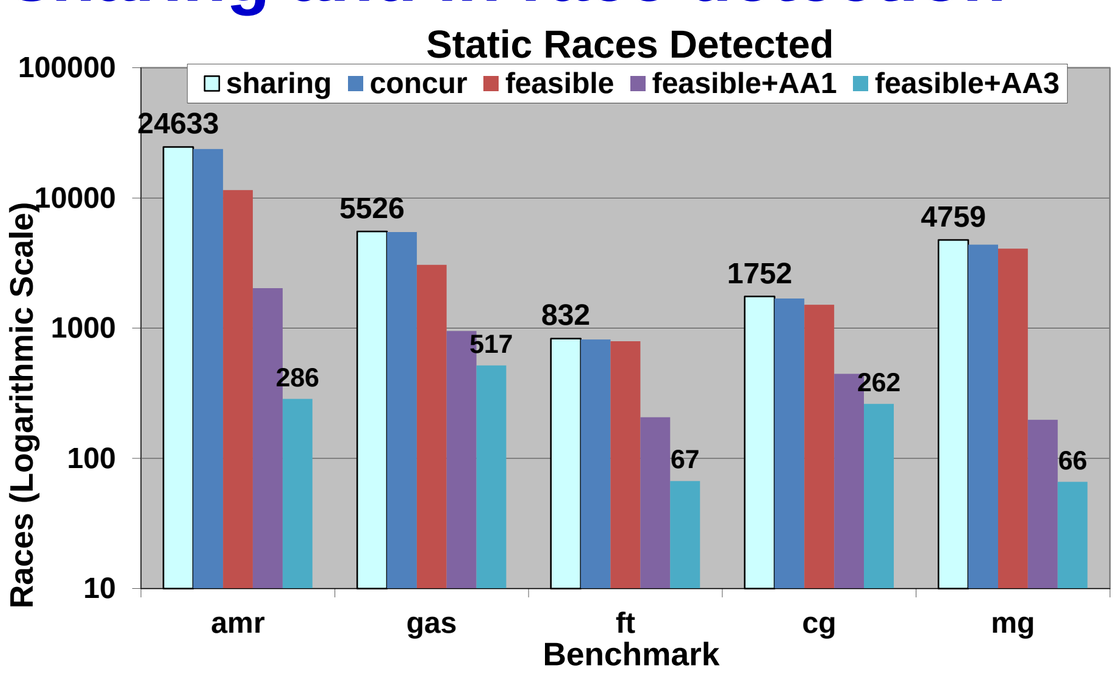
- **Pointer analysis infers maximum reach of each pointer**

```
class Foo {
  Object z;
}
static void bar() {
L1: Foo a = new Foo();
  Foo b = broadcast a from 0;
  Foo c = a;
L2: a.z = new Object();
  Ti.barrier();
  Object d = b.z;
}
```

Points-to Sets		
a	1_p	plocal
b	1_g	global
c	1_p	plocal
1_p.z	2_p	plocal
d	2_g	global

Alocs	1, 2

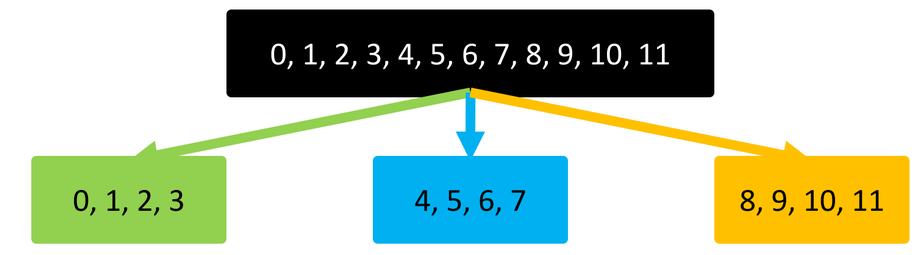
- **Used to infer data locality and sharing and in race detection**



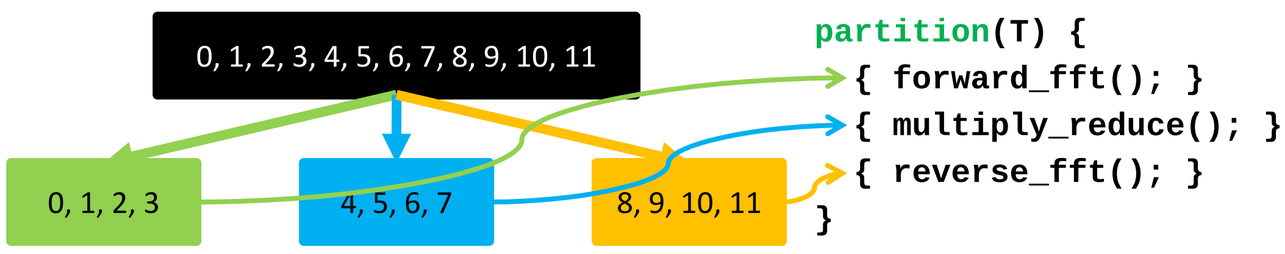
Control Hierarchy

- **Hierarchical *team* data structure used to organize processors**
  - **Library functions provided to facilitate team construction**

```
Team T = new Team();
T.splitTeam(3);
```

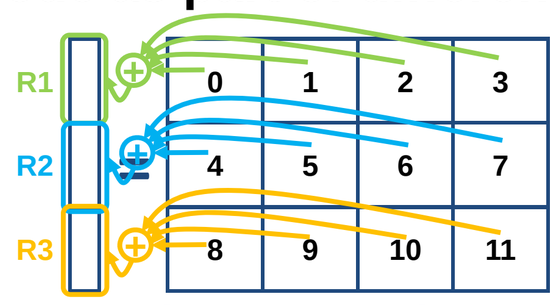


- **Language constructs to move up and down the team hierarchy**
  - **Syntactic scope prevents deadlock**
  - ***Partition* construct allows different teams to execute different code**



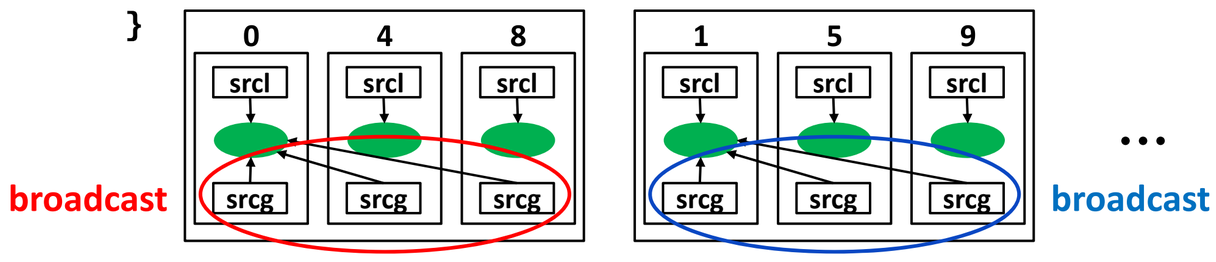
  - ***Teamsplit* construct allows processors to execute same code as part of different teams**

```
teamsplit(T) {
  row_reduce();
}
```



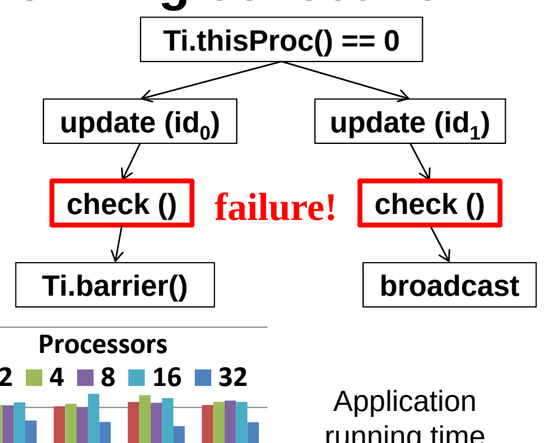
- **Collectives operate over teams**
  - ❖ **Processor indexing is team-relative to allow team-independent code**

```
split(CT) {
  double[1d] srcg = broadcast srcl from 0;
}
```

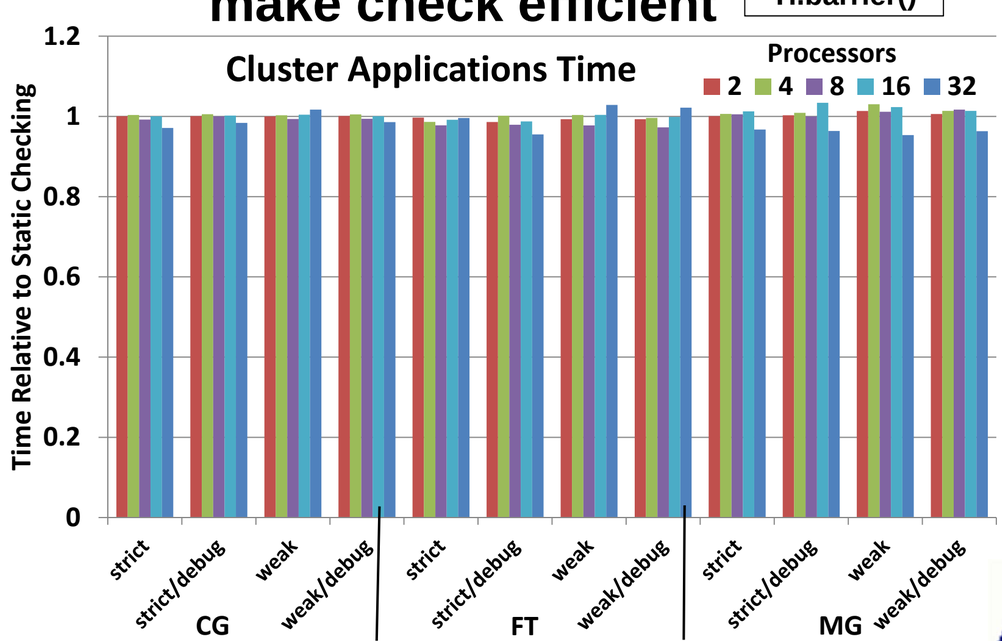


- **Collectives dynamically checked to ensure textual alignment**
  - **Guarantees deadlock freedom**
  - **Runtime keeps track of control flow decisions that affect collective alignment**
  - **Control flow on each processor compared before performing collective**

```
if (Ti.thisProc() == 0)
  Ti.barrier();
int n = broadcast ...;
```



- **Hashing used to make check efficient**



Application running time with four different types of dynamic checking on cluster of dual-processor 2.2GHz Opterons with InfiniBand interconnect

http://titanium.cs.berkeley.edu

The Parallel Computing Laboratory