# Unified Parallel C

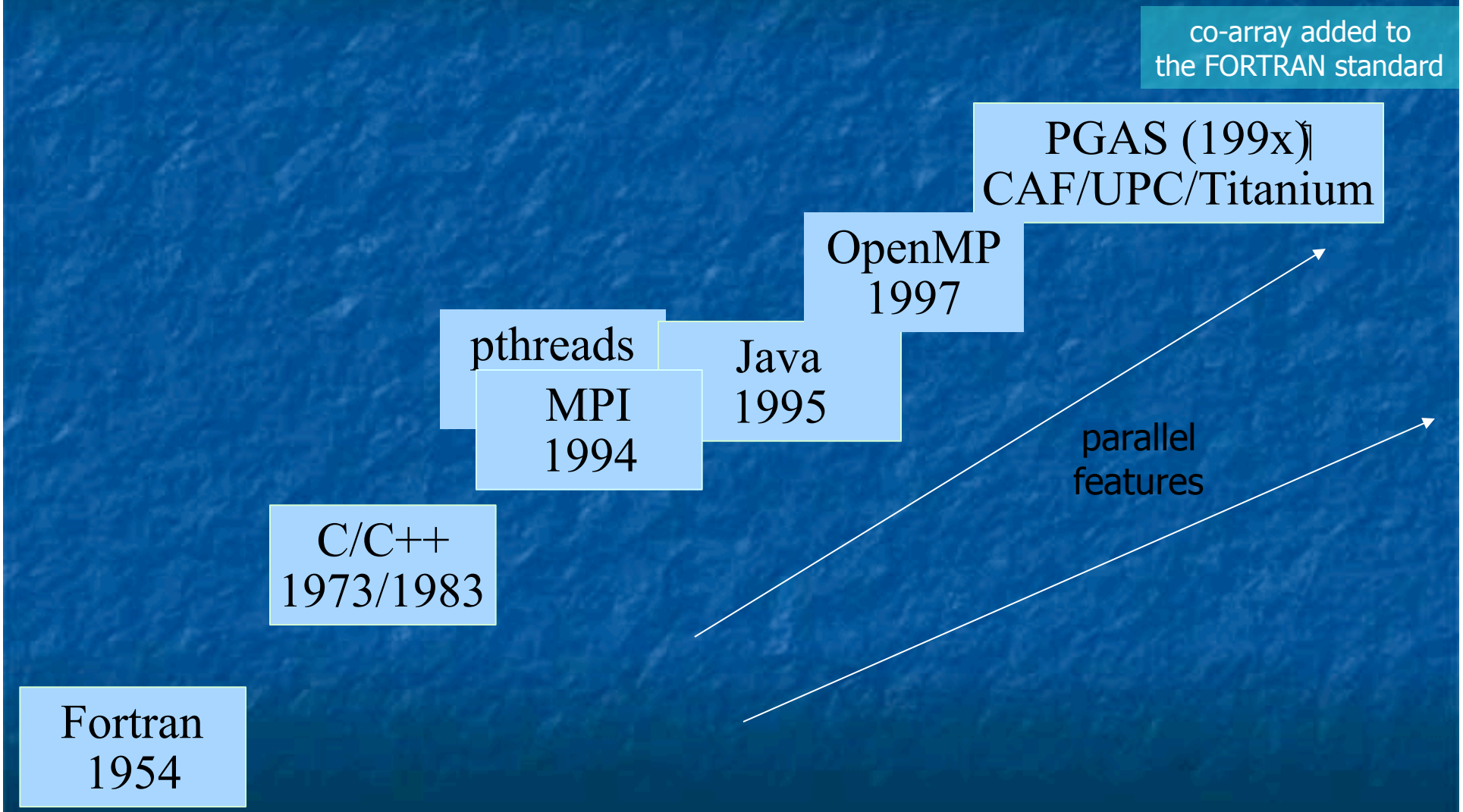
## An overview

SC 22/WG 14 N1374
2009-03-24
Raymond Mak (rmak@ca.ibm.com)

# Agenda

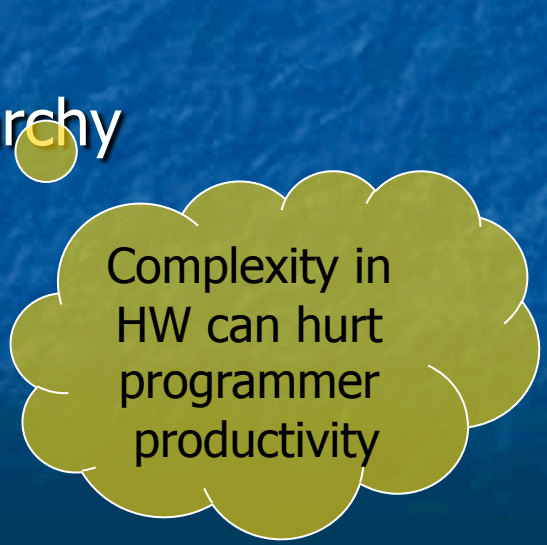
- Productivity, Performance and Parallelism
- Execution models
  - Message Passing
  - Shared Memory
  - Partitioned Global Address Space
- Overview of UPC
- Discussion

# Languages Related to C



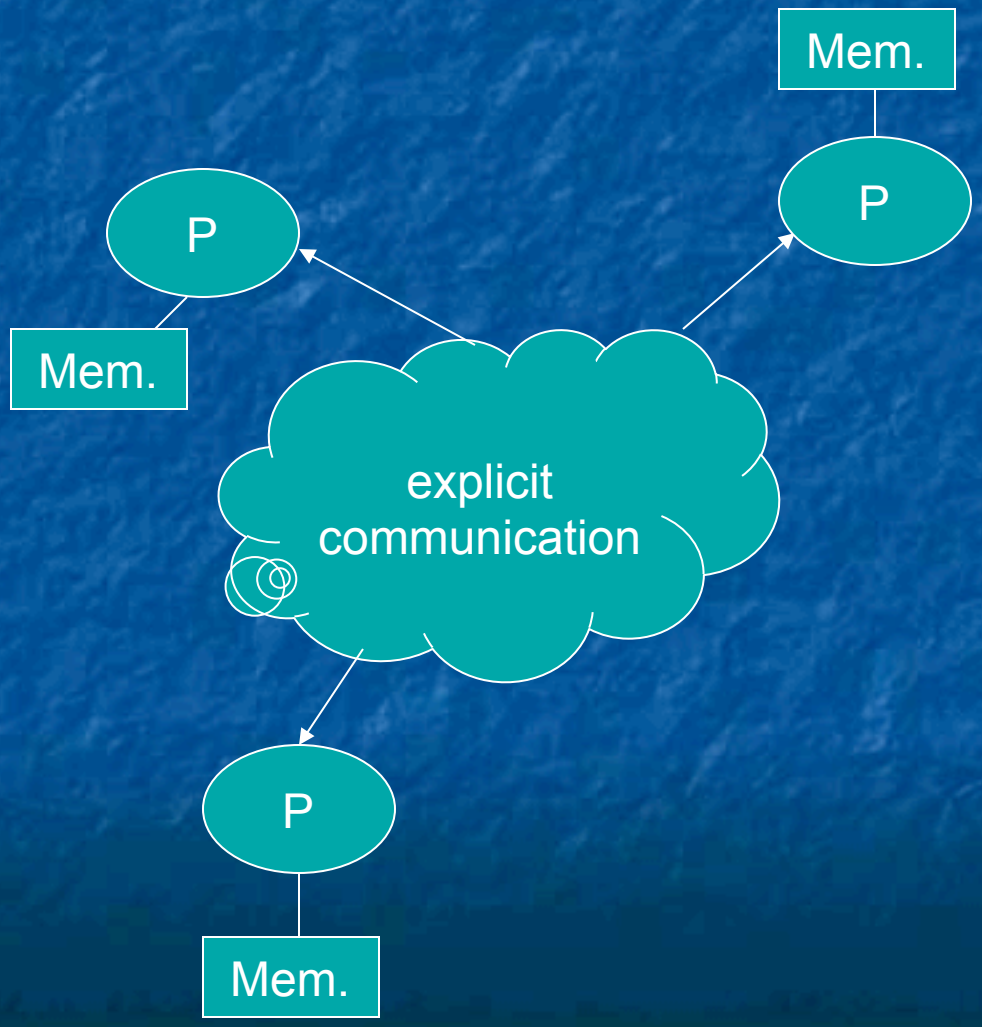
# Productivity -- Ease of Getting Performance

- Single thread performance
  - Clock frequencies leveling off, power limitations
  - Memory is getting farther away
- HW architecture response
  - Multi-core
  - More levels in the memory hierarchy
  - Accelerators
  - Speculation



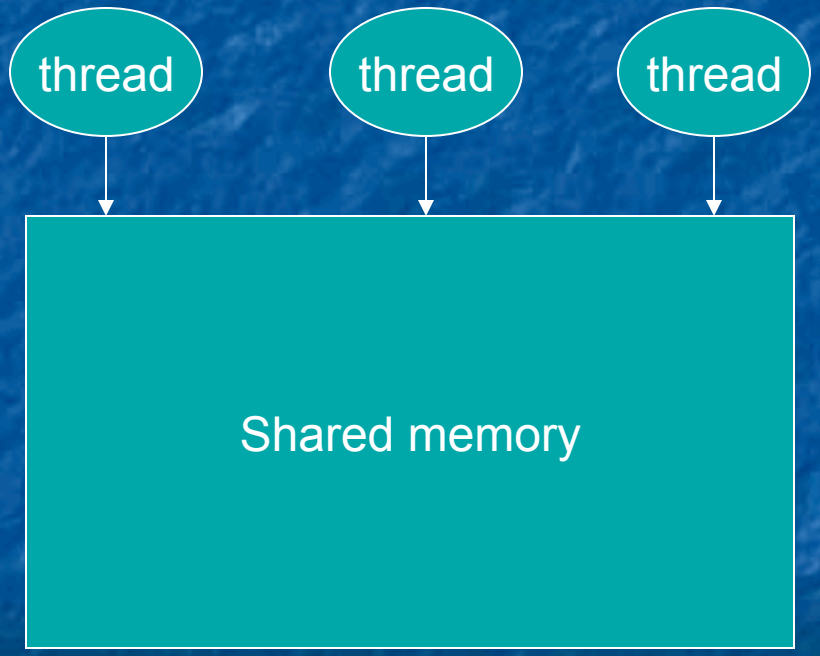
# Execution Models

# Message Passing



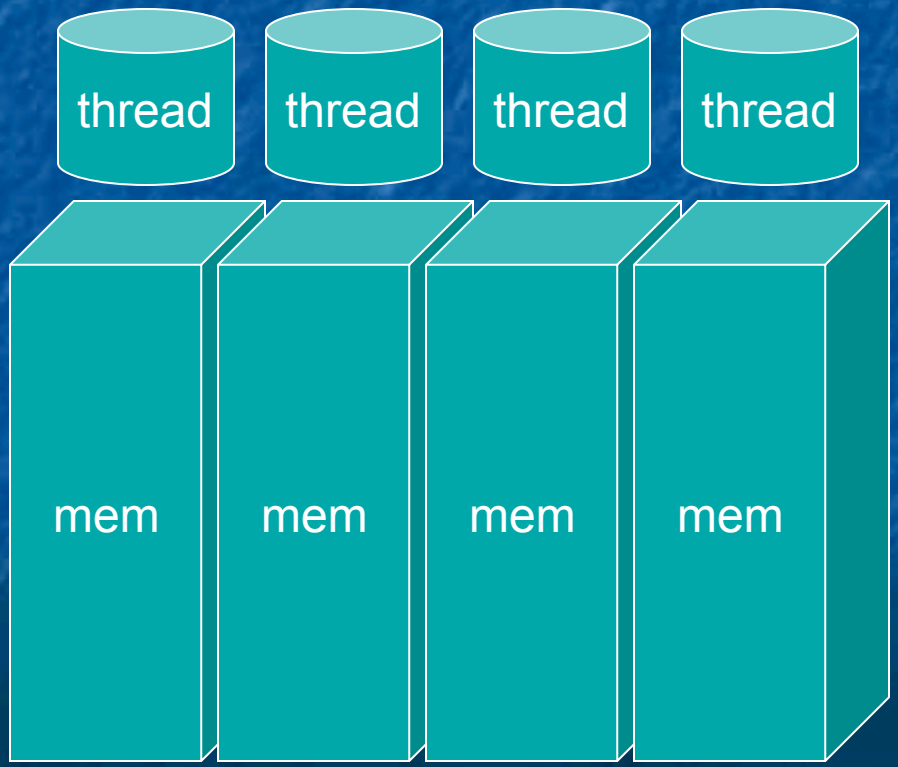

Each processor has local address space. Interact via explicit communication. (MPI)

# Shared Memory




Multiple threads running concurrently.
One address space.
(OMP)

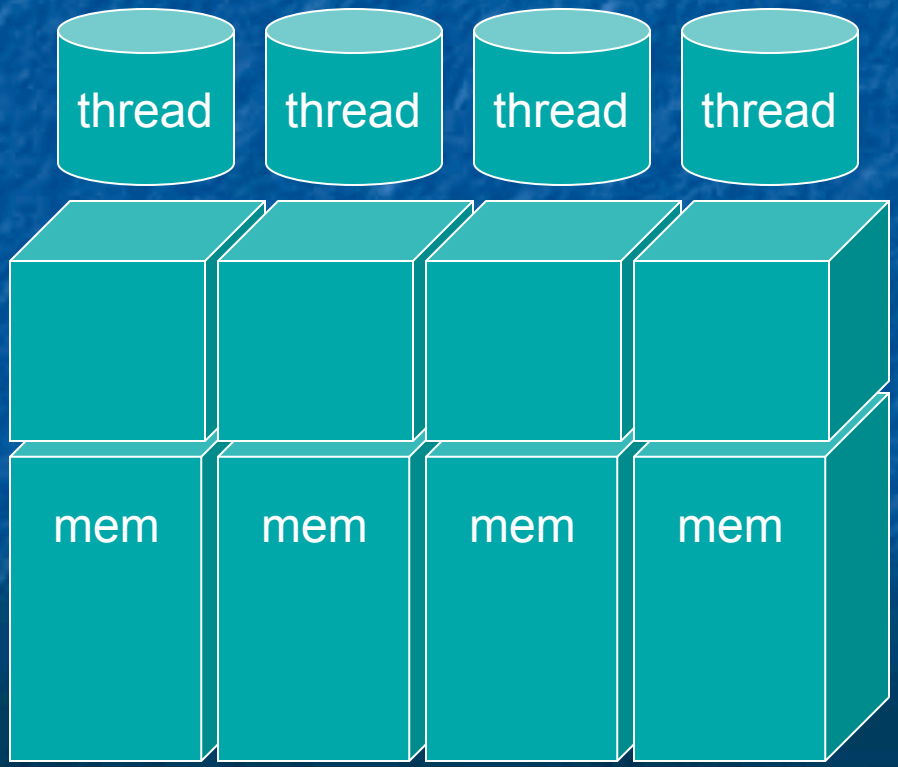
# Partitioned Global Address Space




Multiple threads

Memory accessible by all …

# Partitioned Global Address Space



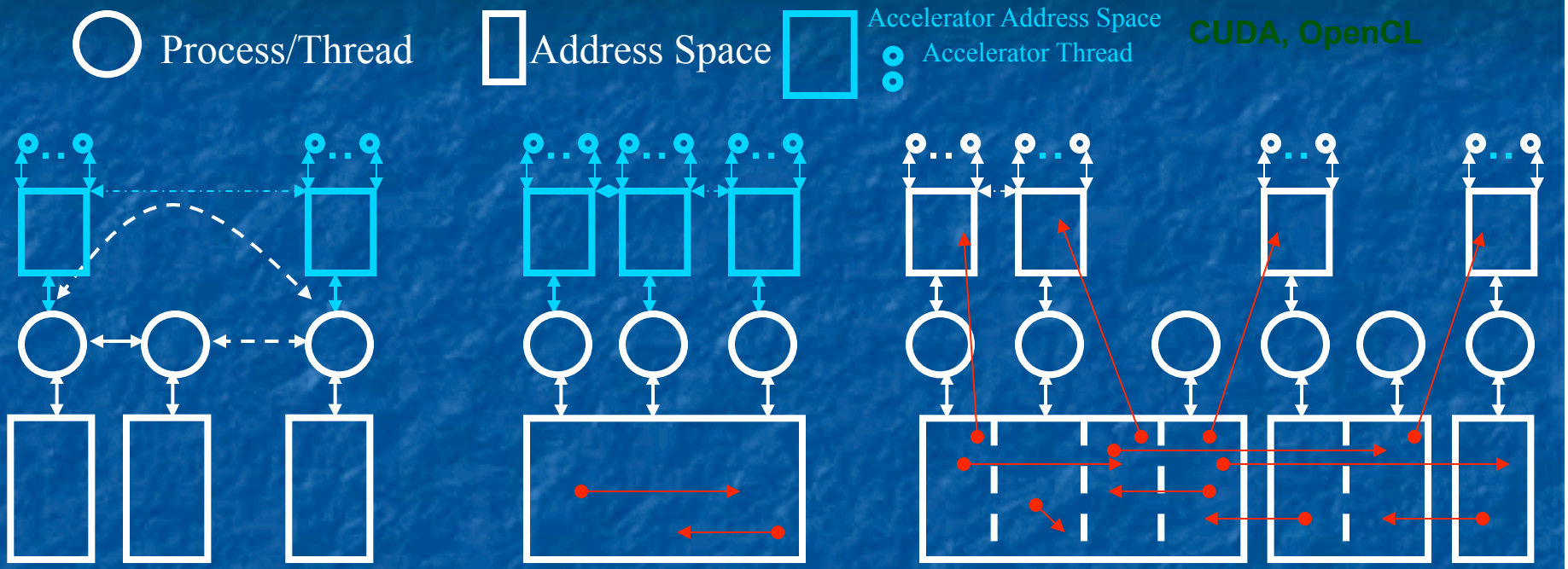

…with affinity

Private to thread

Globally accessible

(UPC)

# Execution Models

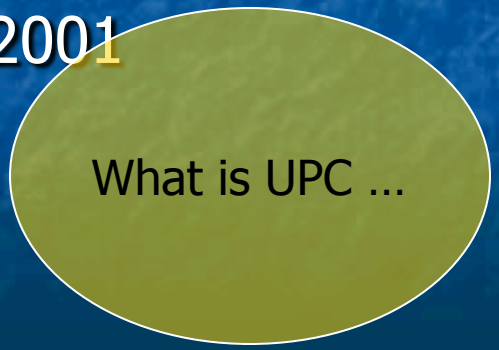



- Computation is performed in multiple places.
- A place contains data that can be operated on remotely.
- Data lives in the place it was created, for its lifetime.

- A datum in one place may reference a datum in another place.
- Data-structures (e.g. arrays) may be distributed across many places.
- Places may have different computational properties

# UPC

- Extension to ISO C
  - A PGAS language
- C's design philosophy
  - Programmer is knowledgeable
  - Minimal language facility to support the right level abstraction, but not too much to hide underlying hardware
    - Close to the hardware when needed
    - Performance without extensive programming effort
    - Code easy to understand and maintenance
- Specification V1.0 completed Feb 2001
- Current specification V1.2
  - http://upc.gwu.edu/
  - http://upc.lbl.gov/publications/



# UPC

intrepid

www.intrepid.com/upc

TotalView

www.etnus.com

Michigan Tech

www.upc.mtu.edu

HP

www.hp.com/go/upc

GCC

CRAY

www.cray.com

Berkeley

upc.lbl.gov

IBM

www.alphaworks.ibm.com/tech/upccompiler

# Quick Overview of the UPC Language

# UPC

- hello word:

```
shared int x;
int y;
int main() {
    printf("hello %d\n", MYTHREAD);
}
```

# UPC

- hello word:

One copy shared by all threads

```
shared int x;
int y;
int main() {
    printf("hello %d\n", MYTHREAD);
}
```

# UPC

- hello word:

```
shared int x;
int y;
int main() {
    printf("hello %d\n", MYTHREAD);
}
```

Private - one per thread

# UPC

- hello word:

A compiler predefined variable

```
shared int x;
int y;
int main() {
    printf("hello %d\n", MYTHREAD);
}
```

# UPC

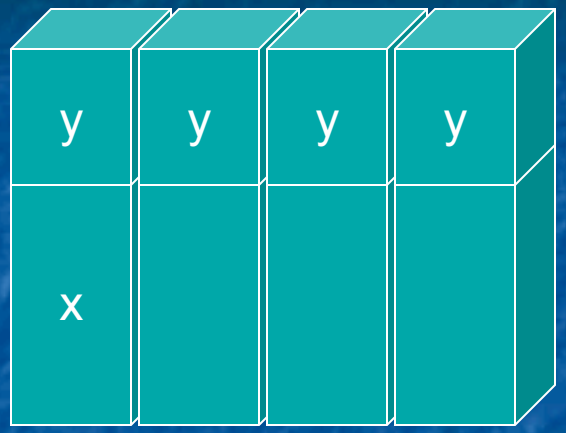
- hello word:

```
shared int x;
int y;
int main() {
    printf("hello %d\n", MYTHREAD);
}
```

```
hello 0
hello 1
hello 2
```

Same code executed by all threads - SPMD

# UPC




- hello word:

```
shared int x;
int y;
int main() {
    printf("hello %d\n", MYTHREAD);
}
```

```
hello 0
hello 1
hello 2
hello 3
```

Same code executed by all threads - SPMD

# UPC

- shared array:

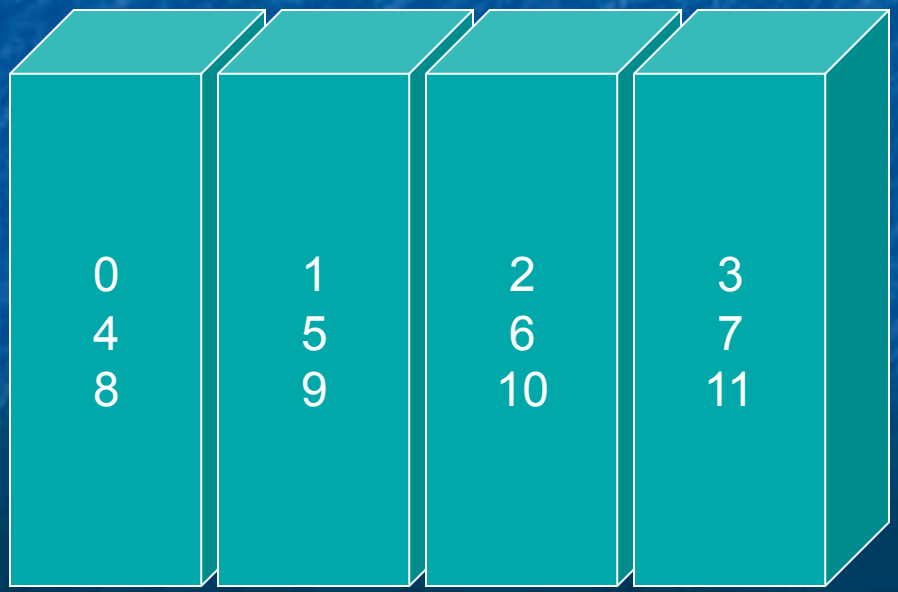
```
shared int arr[ THREADS * 3];
```

Compiler generated var – no. of threads

# UPC

- shared array:

```
shared int arr[ THREADS * 3];
```




All threads can access all elements

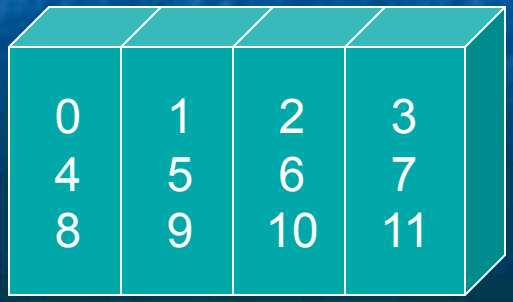
Elements are distributed – with affinity to threads.

# UPC

- shared array:

```
shared int arr[THREADS * 3];
…
upc_forall(i=0; i<THREADS*3; ++i; i)
    arr[i] = 0;
```

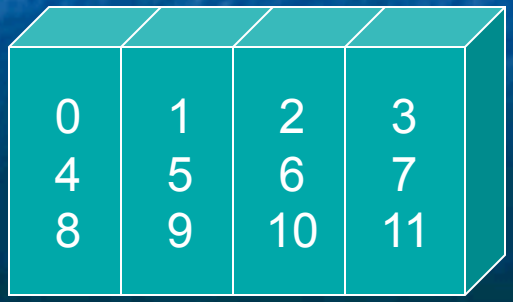
Affinity expression



# UPC

- shared array:

```
shared int arr[THREADS * 3];
…
upc_forall(i=0; i<THREADS*3; ++i; i) {
    if (i%THREADS == MYTHREAD)
        arr[i] = 0;
}
```



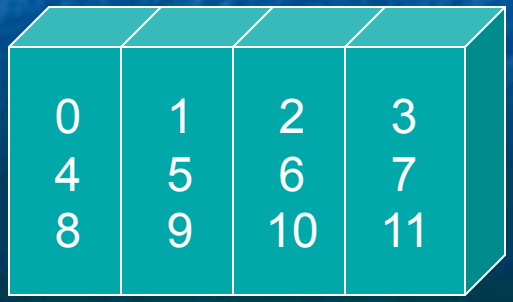

Transform to a for loop

# UPC

- shared array:

```
shared int arr[THREADS * 3];
…
upc_forall(i=0; i<THREADS*3; ++i; &arr[i])
    arr[i] = 0;
```

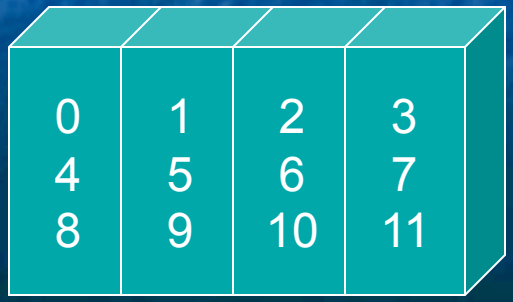
threadof arr[i]



# UPC

- shared array:

```
shared int arr[THREADS * 3];
…
upc_forall(i=0; i<THREADS*3; ++i; &arr[i])
    if (upc_threadof(&arr[i] == MYTHREAD)
        arr[i] = 0;
```




Transform to a for loop

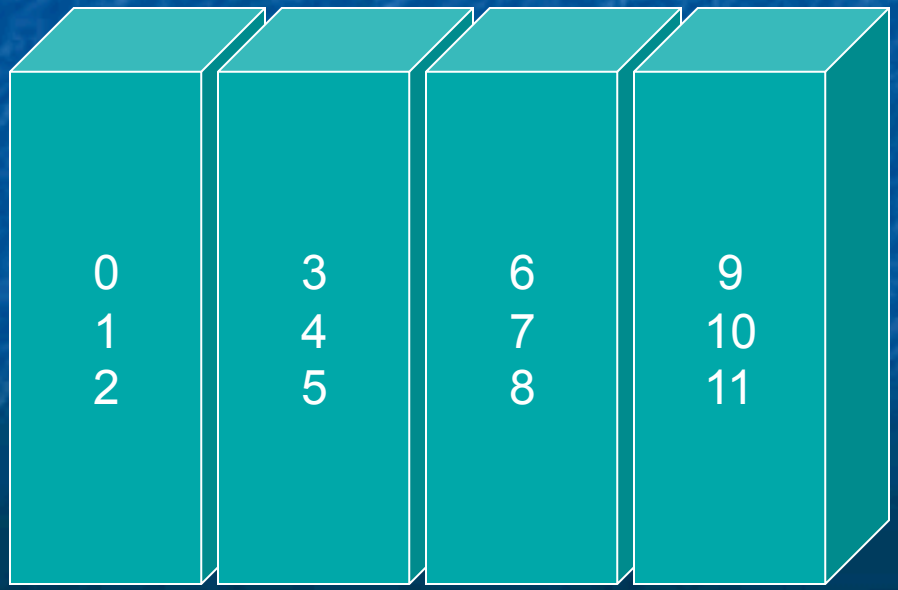
# UPC

- shared array:

block size

control on how the array is distributed

```
shared[3] int arr[ THREADS * 3];
```




elements are distributed by blocks of 3

# UPC

- Shared pointers

```
shared int *p;
shared int * shared p;
```

- Memory management
  - upc_global_alloc, upc_local_alloc, upc_free
- Synchronization
  - upc_lock, upc_unlock
  - upc_barrier, upc_fence, upc_wait, upc_notify
- Utility functions
  - upc_memcpy, upc_memput, upc_memget, upc_memset
- Memory consistency model
  - Strict/relaxed

# PGAS languages

- Features
  - Small set of data parallel primitives
    typically
    g
    rafted on an existing language: Co-Array Fortran, UPC, Titanium
  - Shared memory-like programming with locality awareness –
    shared data is
    e
    xplicitly declared and distributions are implicit in the declaration
  - SPMD threading
    m
    odel with synchronization primitives (barriers, fences, and locks)
  - Collective communication and parallel I/O through libraries
- Implementation
  - Can be mapped to shared
    m
    emory, distributed memory and combinations (clusters of SMPs)
    One-sided communication

# Discussions

# Discussion

- Given the trend in hardware architecture, should C add features to support parallel programming ?
- From C's perspective, is something like UPC attacking the problem at the right level and scope ?
- What should be the role of the language standard ?
- How could the C committee be involved …
  - Adviser to the UPC Working groups ?
  - Study group within the C committee ?
  - Technical report ?