Unified **P**arallel **C**
UPC

Slides based on those found at http://www2.hpcl.gwu.edu/pgas09/tutorials/upc_tut.pdf

see upc.gwu.edu for more UPC information, or the Berkeley site

Contrast with MPI

	MPI	OpenMP	UPC
Programming Model	Message passing	Shared Memory	Dist. Shared Memory
Expressing Parallelism	Library	Library Directives	C Extension
Architecture supported	MIMD	SMP	MIMD
Incremental parallelizing	Not really	Yes	Not really
Locality exploitation	Yes, using dist and comm	NO	Yes: blocking and affinity
Collective operations	Yes	Compiler/Pgmmer input	Compiler/Pgmmer Input
Data distribution in Declarations	No	N/A	Yes
Memory Consistency	N/A	Strict	Strict or relaxed
Dynamic memory alloc	Private only	Yes	Private or shared, w/or without blocking
pointers to dist. data	N/A	N/A	Yes
Synchronization	Barriers/Wait	critical sect, locks, etc.	barriers, locks, cons. ctl

UPC -- design philosophy

- Start with C
- Keep C low-level control features (addresses, pointers, etc.)
- Add parallelism, learn from previous languages
- Take input and suggestions from the developer's community
- Integrate and work with experts from government, vendors, academia

UPC design philosophy

- Assume programmers know what they are doing
- Put programmers close to the hardware, let them exploit hardware properties
  - Can get good performance without super-powerful compilers
  - Can also get into trouble
- Concise and efficient syntax like C
- Easy to implement on different architectures
- High performance at the system and node levels

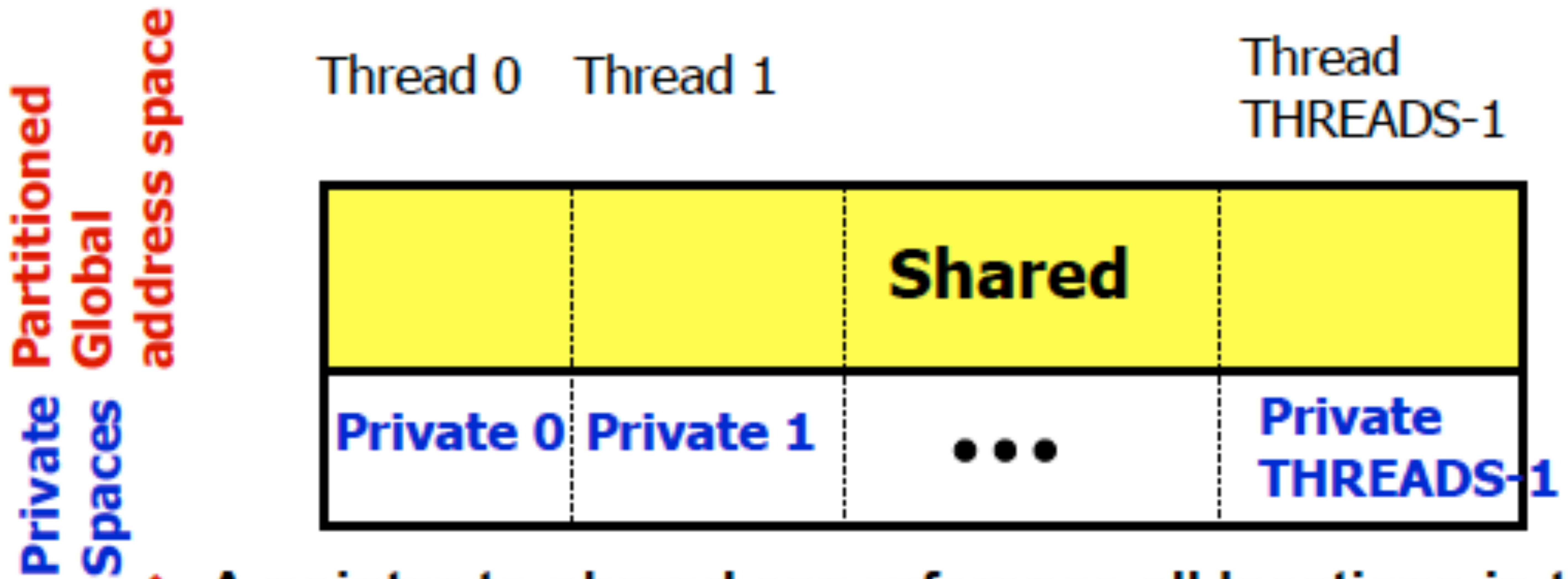
UPC a PGAS language

- PGAS is Partitioned Global Address Space
- Unlike MPI, one address space covers all objects ("*objects*" in the sense of something in memory, not in the OO sense)
- Unlike OpenMP/Pthreads/Java, address space is assumed to be partitioned across multiple nodes and (perhaps) physically disjoint address spaces

UPC Execution Model

- A number of threads working independently in a SPMD fashion
  - MYTHREAD specifies thread index (0..THREADS-1)
  - Number of threads specified at compile-time or run-time
- Synchronization when needed
  - Barriers
  - Locks
  - Memory consistency control

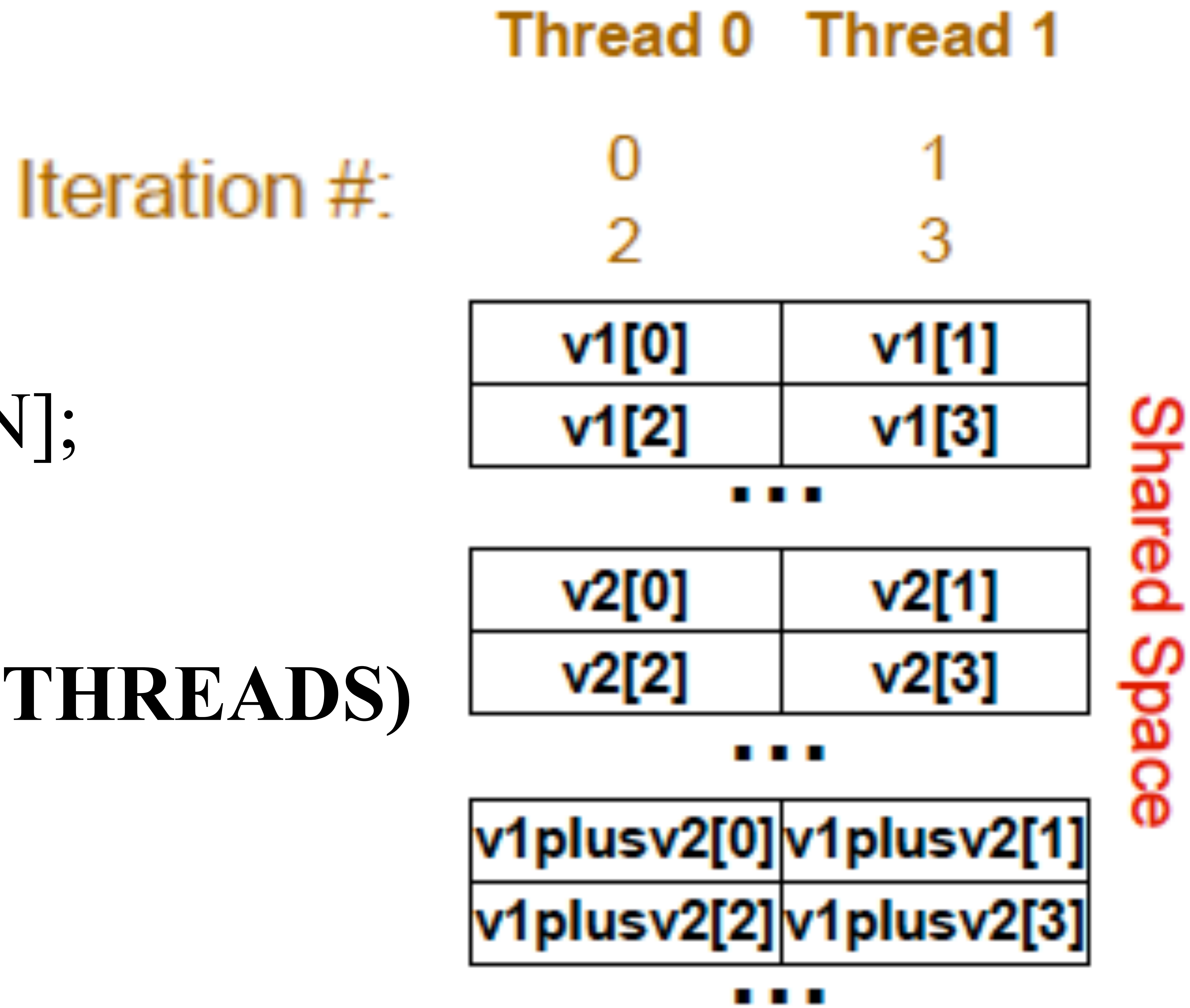
UPC Memory Model



- **A pointer-to-shared can reference all locations in the shared space, but there is data-thread affinity**
- **A private pointer may reference addresses in its private space or its local portion of the shared space**
- **Static and dynamic memory allocations are supported for both shared and private memory**

Vector add in UPC

```
#include <upc_relaxed.h>
#define N 100*THREADS

shared int v1[N], v2[N], v1plusv2[N];
void main( ) {
   int i;
   for (i=MYTHREAD; i < N; I+= THREADS)
       v1plusv2[i] = v1[i] + v2[i];
}
```



A better implementation using upc_forall

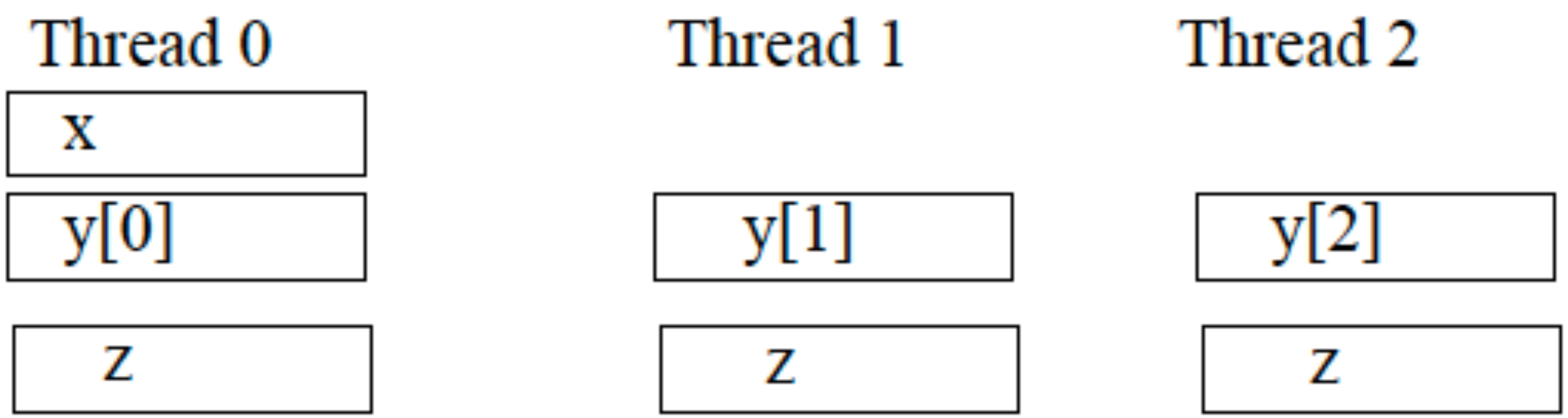
```
#include <upc_relaxed.h>
#define N 100*THREADS

shared int v1[N], v2[N], v1plusv2[N];
void main( ) {
   int i;
   upc_forall (i=0; i < N; i++;)
       v1plusv2[i] = v1[i] + v2[i];
}
```

- upc_forall handles distribution of work
- Easier since UPC keeps track of assignment of work to threads

Shared and private data in UPC

Assume THREADS=3

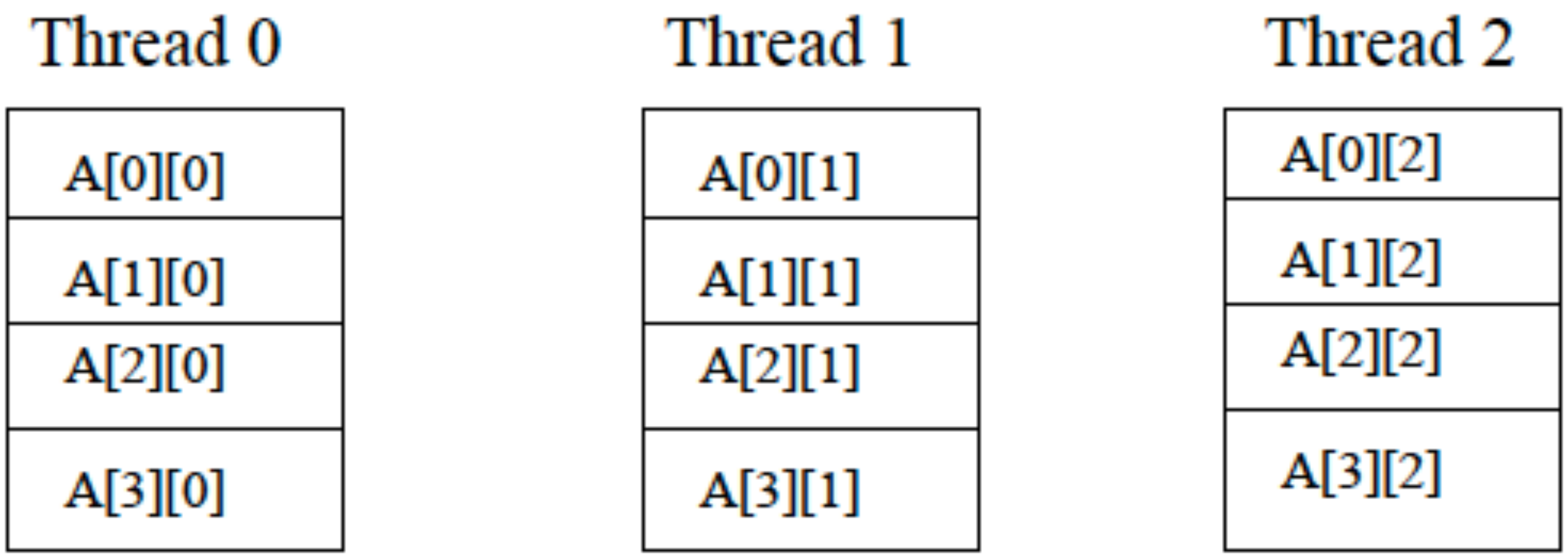
shared int x; /* x has an affinity for process 0 */
share int y[THREADS]
int z; **// private -- one per thread**



Shared and private data (2)

shared int A[4][THREADS]

Data is spread across the processors in a round-robin fashion



Different distributions of data

- Default block size is 1 -- gives a cyclic distribution
- Other block sizes can be specified to achieve a block or block-cyclic distribution
- Thread affinity is given by $\left\lfloor \frac{i}{blocksize} \right\rfloor \bmod THREADS$

shared [block-size] type array-name[N]

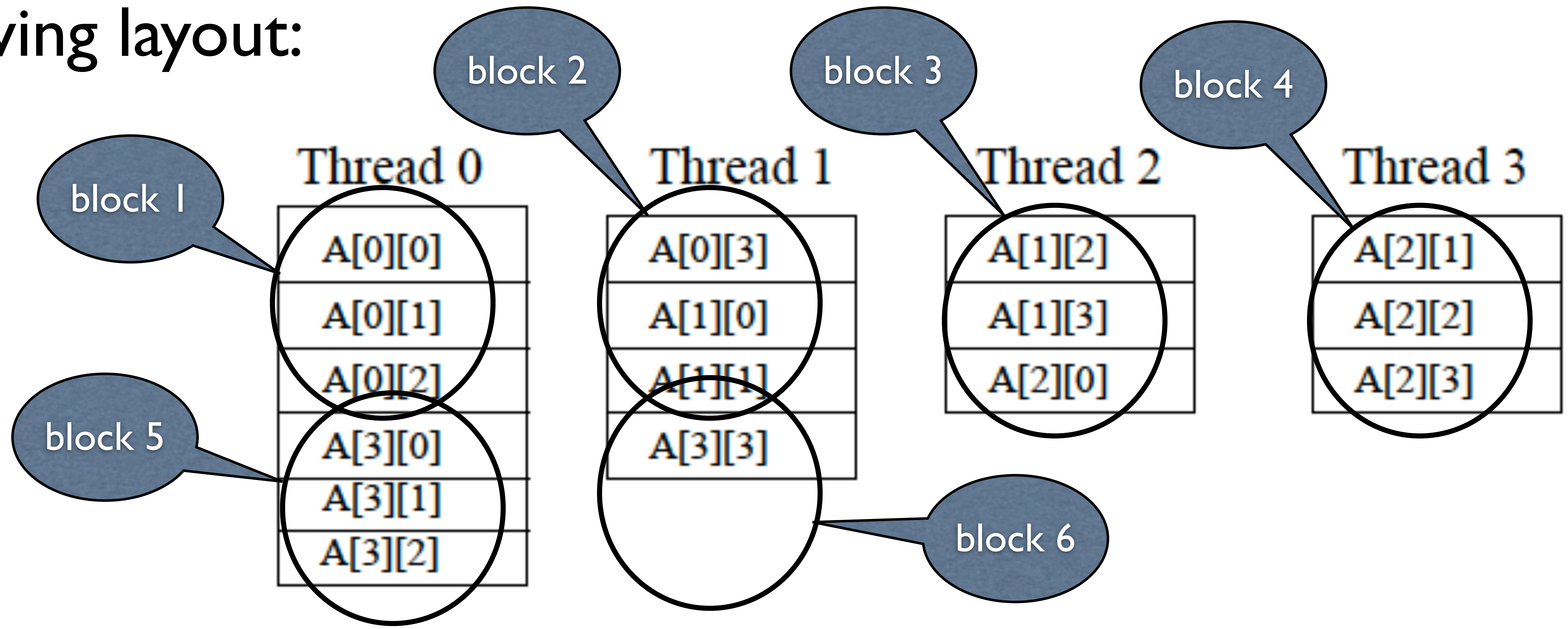
shared[4] int a[16]

Shared and private data

- Assume **THREADS = 4**
- shared [3] int A[4][THREADS]
- will result in the following layout:

UPC provided string functions can be used to move blocks of data among threads

Query operators provided to determine blocking information and affinity information of arrays and types



UPC Pointers

- Two attributes
  - location of storage pointed to
  - location of the pointer itself

Where pointer is

Where pointed to data is

data location/ pointer location	private	shared
private	PP	PS
shared	SP	SS

UPC Pointers

int *p1; /* private pointer pointing to local storage */

shared int *p2; /* private pointer pointing into shared space. *A "pointer to shared"* */

int *shared p3; /* shared pointer pointing locally */ ← a bad idea. Why?

shared int* shared p4; /*shared pointer pointing into the shared space */

People sometimes use *shared pointer* to mean a pointer pointing into the shared space (p2), but could also be a pointer residing in the shared space (p4)

UPC Pointers

private pointer (no *shared*)

private int (no *shared*)

int *p1; /* private pointer pointing to local storage */

shared int *p2; /* private pointer pointing into shared space. *A "pointer to shared"* */

shared ptr

int *shared p3; /* shared pointer pointing locally */

a bad idea. Why?

shared int* shared p4; /*shared pointer pointing into the shared space */

shared int

shared ptr

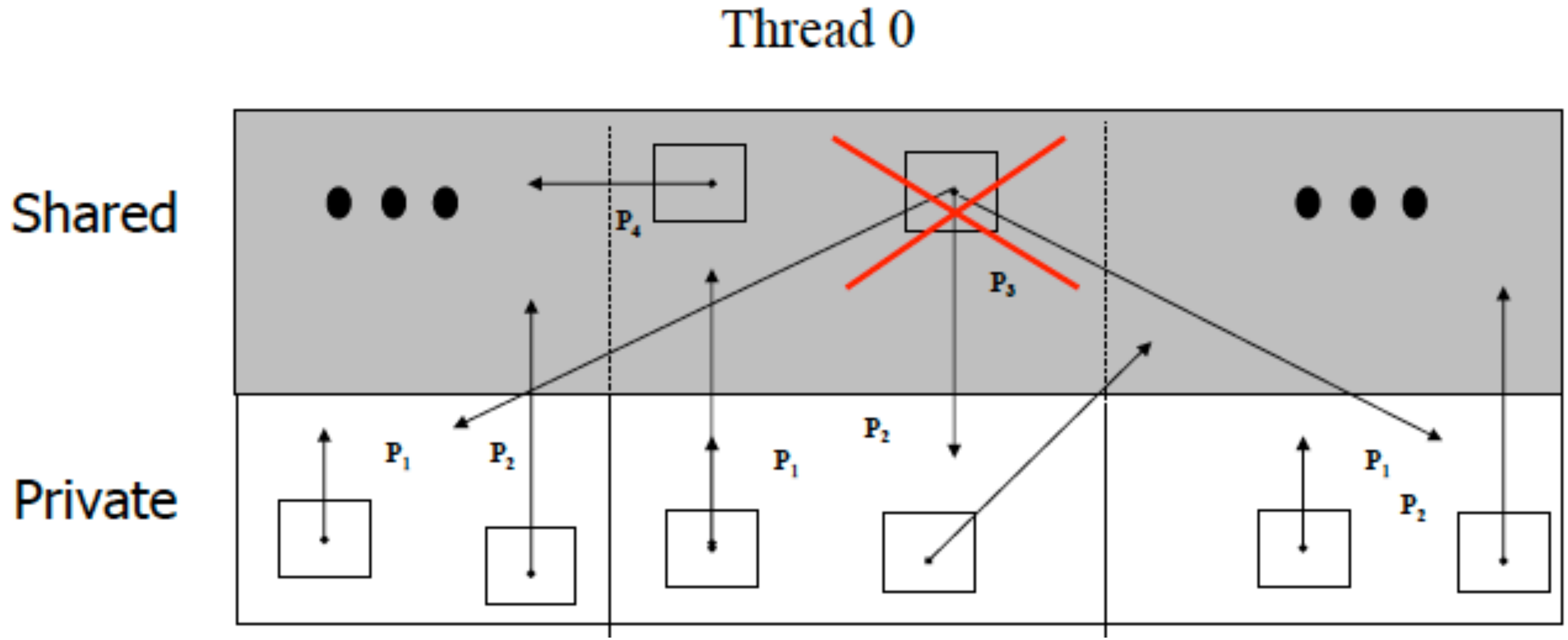
UPC Pointers

int *p1; /* private point pointing to local storage */

shared int *p2; /* private point pointing into shared space */

int *shared p3; /* shared pointer pointing locally */

shared int *shared p4; /*shared pointer pointing into the shared space */



Implementation of pointer to shared objects

- UPC pointers to shared objects have three fields:
  - **thread number:** thread whose storage that contains to the object being pointed to
  - **Block address:** local address of the block that contains the object
  - **Phase:** contains the location of the object within the block

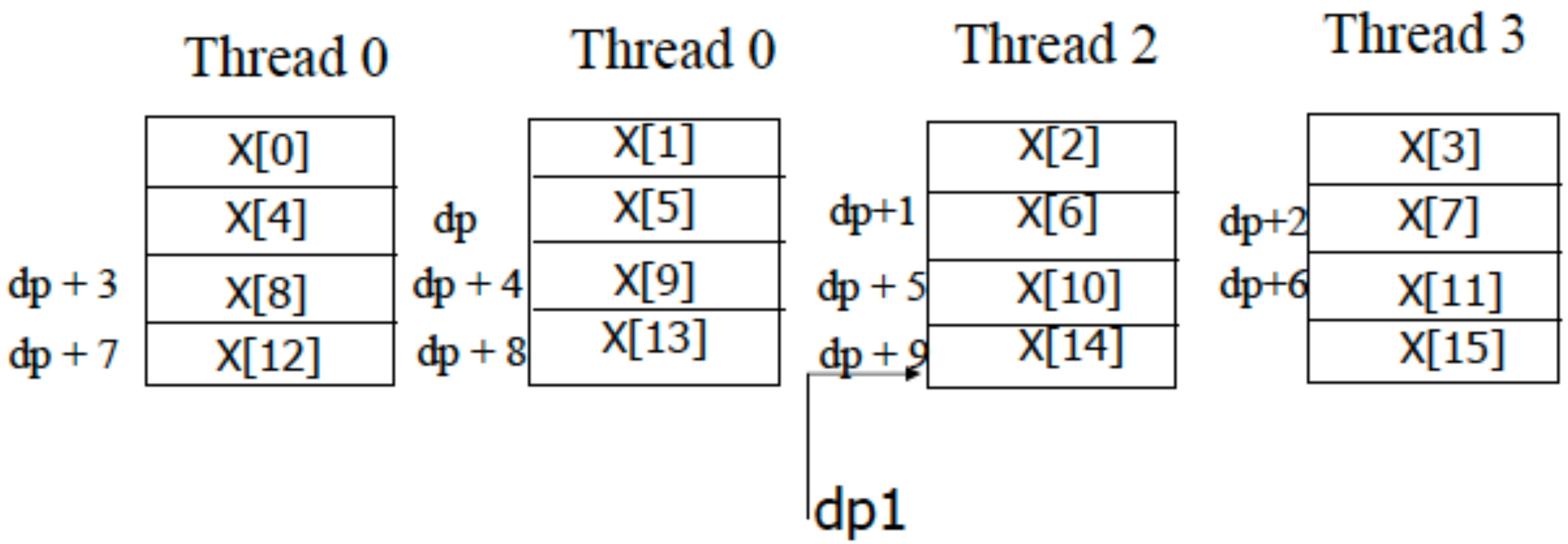
UPC Pointers manipulations

- Pointer arithmetic supports blocked and non-blocked array distributions
- Casting of shared to private is legal, ***but not vice versa***
- Casting from pointer-to-shared to pointer-to-private may lose the "owning" thread number information
- Casting from pointer-to-shared to pointer-to-private is well defined ***only*** if the private pointer resides on the same thread as the data

UPC pointer arithmetic

Assume **THREADS=4**

```
#define N 16
shared int x[N];
shared int *dp=&x[5], *dp1;
dp1 = dp+9;
```



blocking is part of type -- can lead to interesting pointer arithmetic

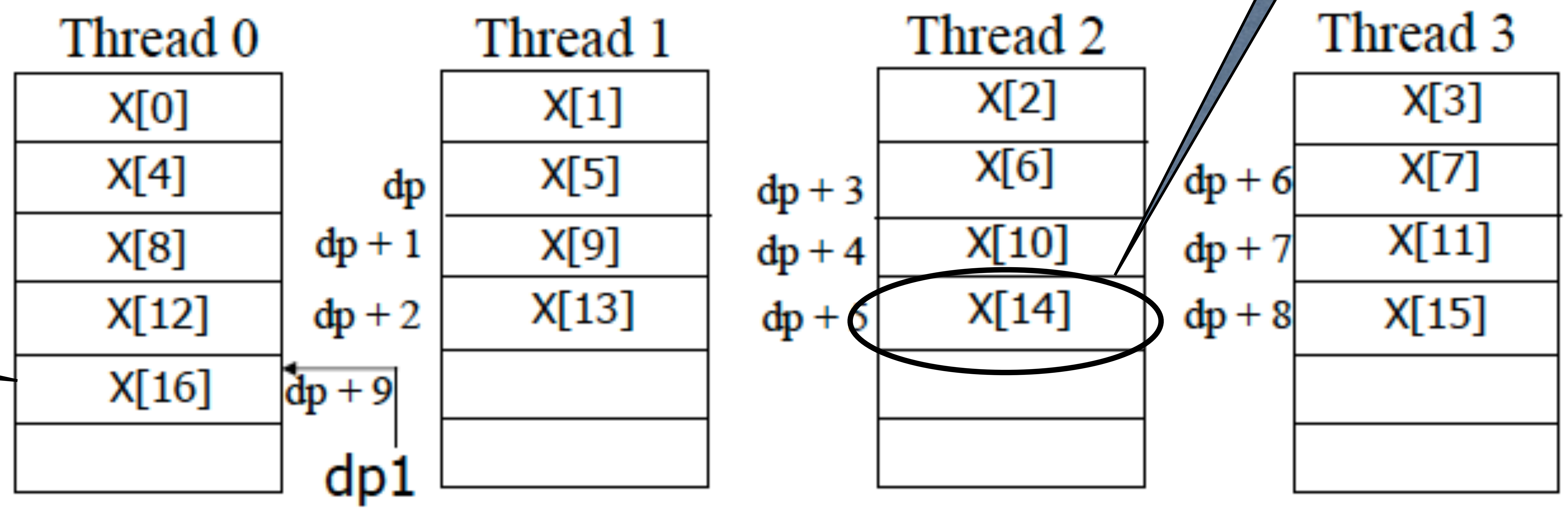
Assume **THREADS=4**

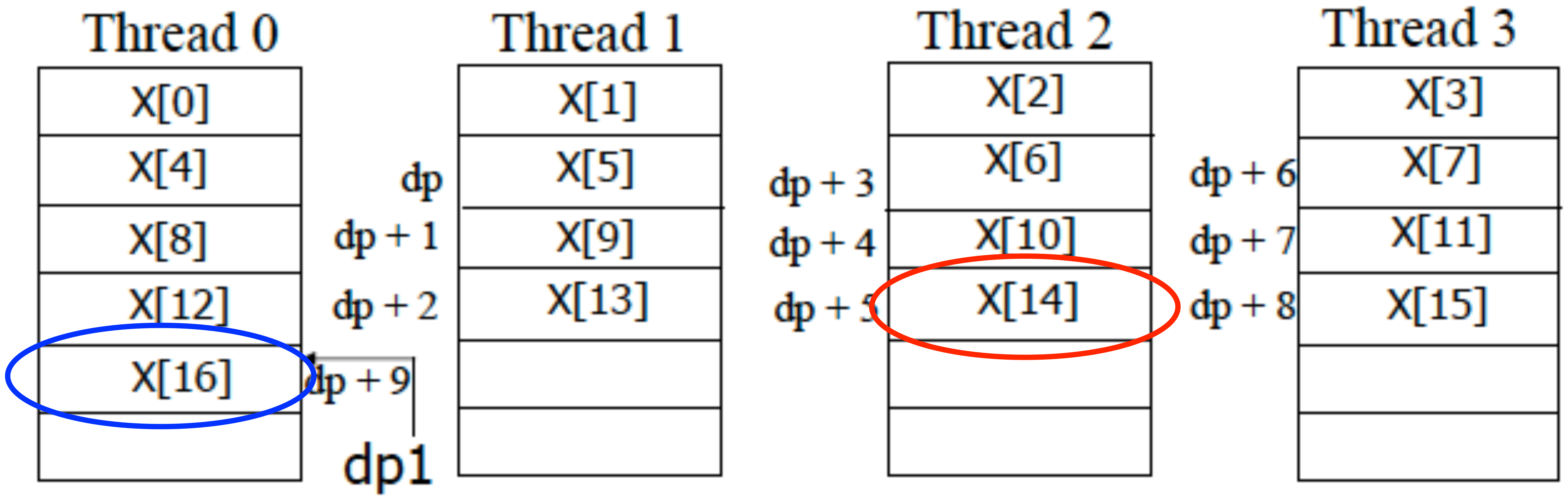
```
shared int x[N];
shared[3] int *dp = &x[5], *dp1;
dp1 = dp + 9;
```

Pointer declared with blocksize of 3 points to array with default blocksize of 1

9 elements past by array index value

9 elements past assuming blocks of size 3





- Allows pointer arithmetic to be used to scan elements within a block and to exploit locality

- Pointer follows its own declared blocking and not that of what it points to

```
shared int x[N];
shared[3] int *dp = &x[5], *dp1;
```

More pointer fun

- Given the declarations

shared[3] int *p;

shared[5] int *q;

- Then

p=q is acceptable (may need an explicit cast w/some implementations)

- Pointer p, however, will follow pointer arithmetic for blocks of 3, not 5

UPC forall

- Distributes work across threads
- Simple C-like syntax and semantics
- upc_forall(init; test; incr; affinity);
  - affinity can be an integer expression or
  - a reference to (address of) a shared object

Exploiting locality with upc_forall

Example 1:

```
shared int a[100], b[100], c[100];
int i;
upc_forall (i=0; i < 100; i++; &a[i])
   a[i] = b[i]*c[i];
```

Iteration *i* executes on the processor that a[i] resides on

Example 2:

```
shared int a[100], b[100], c[100];
int i;
upc_forall (i=0; i < 100; i++; i)
   a[i] = b[i]*c[i];
```

expression mod THREADS gives the thread iteration executes on. Same distribution as in example 1.

More working sharing with upc_forall

Example 3: distribute by chunks

```
shared int a[100], 1[100], a[100]
int i;
upc_forall (i=0; i < 100; i++; (i*THREADS)/100)
   a[i] = b[i] * c[i];
```

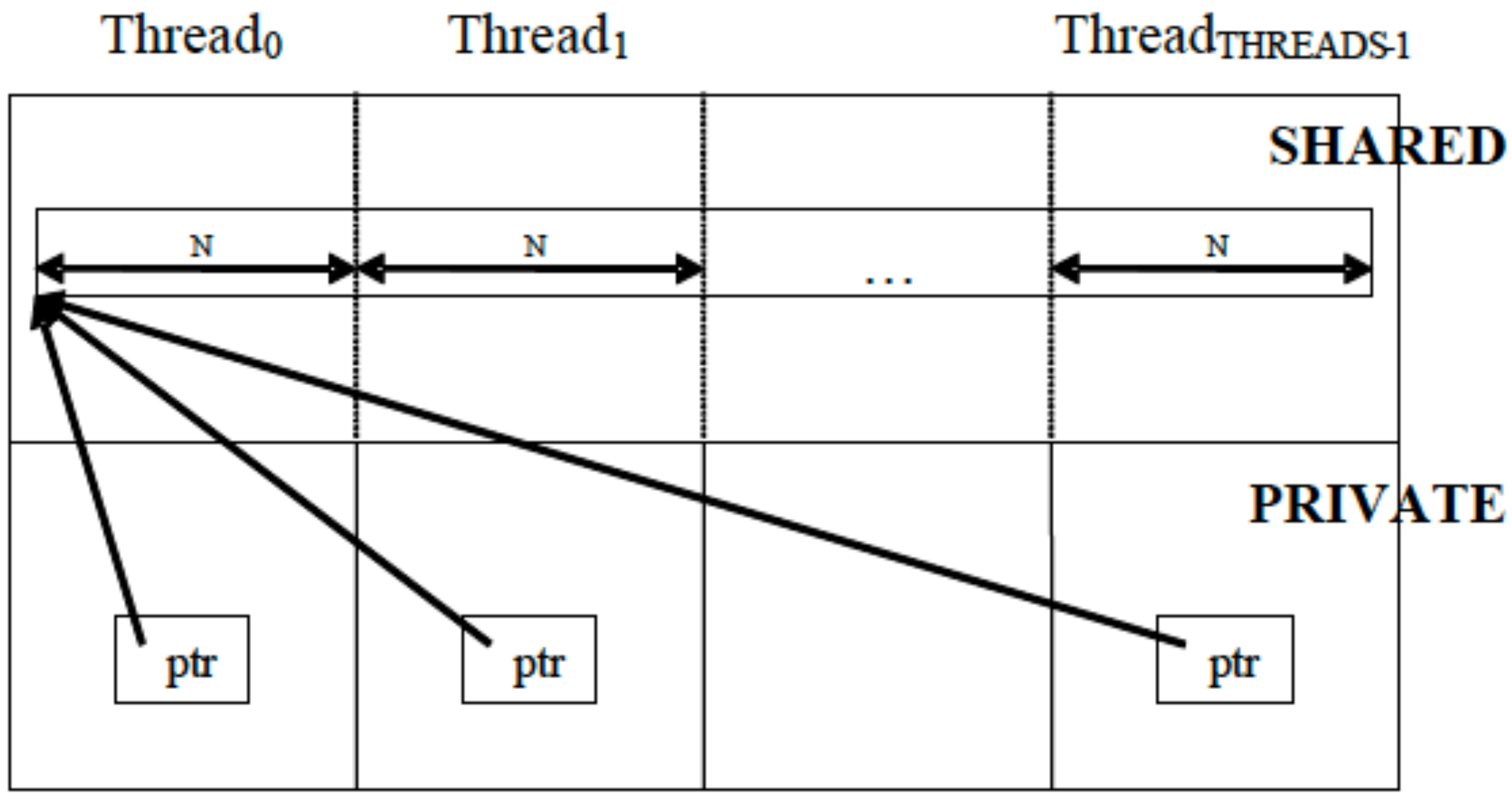
i	i*THREADS	i*THREADS/100
0..24	0..96	0
25..49	100..196	1
50..74	200..296	2
75..99	300..396	3

Other supported functionality

- Dynamic memory allocation
- Synchronization
- Memory consistency models

Dynamic memory allocation

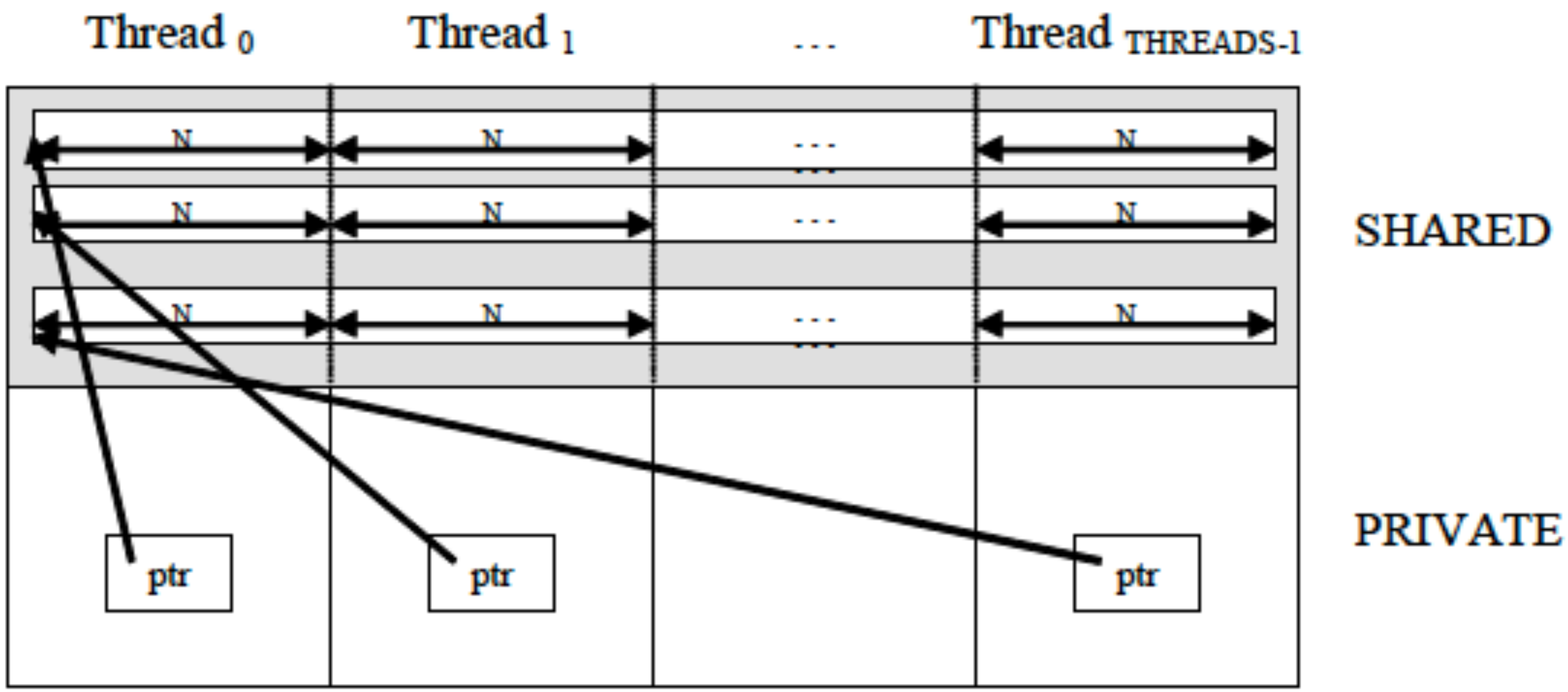
- *Collective operations* executed by every thread, allocates a contiguous chunk in the shared space, all threads get the same value



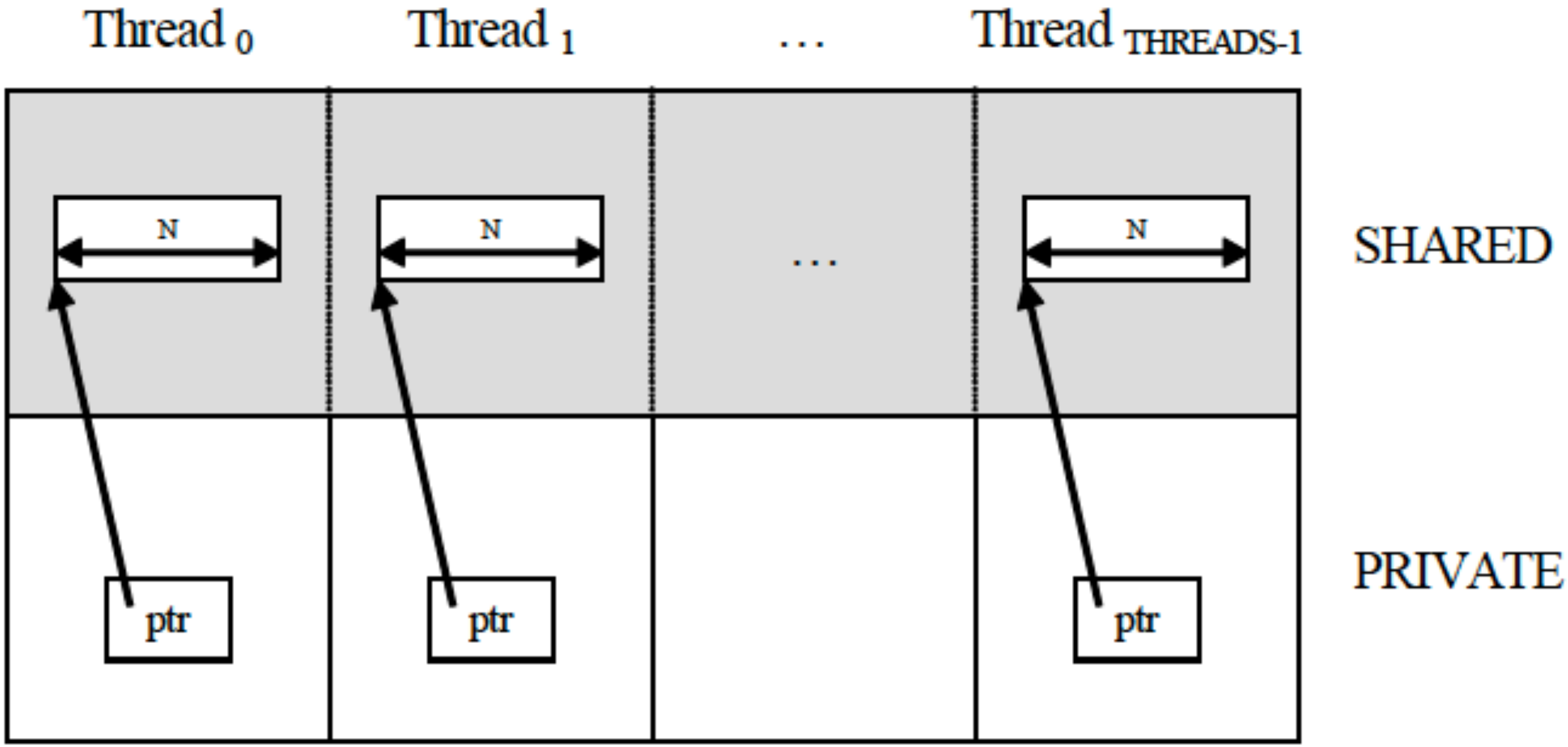

```
shared [N] int *ptr;
ptr = (shared [N] int *)
        upc_all_alloc( THREADS, N*sizeof( int ) );
```

- *non-collective operations* allocate a contiguous region in the shared space
- each thread invoking this allocates a different region and gets a pointer to that region

```
shared [N] int *ptr;

ptr =
   (shared [N] int *)
   upc_global_alloc( THREADS, N*sizeof( int ));
```



Local shared memory allocation




```
shared [] int *ptr;
ptr = (shared [] int *)upc_alloc(N*sizeof( int ));
```

UPC free

- UPC_free frees up storage **pointed-to** by a shared pointer
- local allocations to local pointers can be done by malloc

UPC synchronization

- barriers that can involve various subsets of threads
- Can be blocking or non-blocking
  - With non-blocking, hit the barrier, do some work, then wait for the barrier when data written before the barrier is needed
- Locks, lock_attempt allows a lock to be checked

Memory models

- Can either have a relaxed or strict memory model
- Strict is SC
- Can be specified on a per-variable basis
- The compiler enforces this

Reduced Coding Effort is Not Limited to Random Access– NPB Examples

		SEQ1	UPC	SEQ2	MPI	UPC Effort (%)	MPI Effort (%)
NPB-CG	#lines	665	710	506	1046	6.77	106.72
	#chars	16145	17200	16485	37501	6.53	127.49
NPB-EP	#lines	127	183	130	181	44.09	36.23
	#chars	2868	4117	4741	6567	43.55	38.52
NPB-FT	#lines	575	1018	665	1278	77.04	92.18
	#chars	13090	21672	22188	44348	65.56	99.87
NPB-IS	#lines	353	528	353	627	49.58	77.62
	#chars	7273	13114	7273	13324	80.31	83.20
NPB-MG	#lines	610	866	885	1613	41.97	82.26
	#chars	14830	21990	27129	50497	48.28	86.14

$$UPC_{effort} = \frac{\#UPC - \#SEQ1}{\#SEQ1} \qquad MPI_{effort} = \frac{\#MPI - \#SEQ2}{\#SEQ2}$$

SEQ1 is C
SEQ2 is from NAS, all FORTRAN except for IS