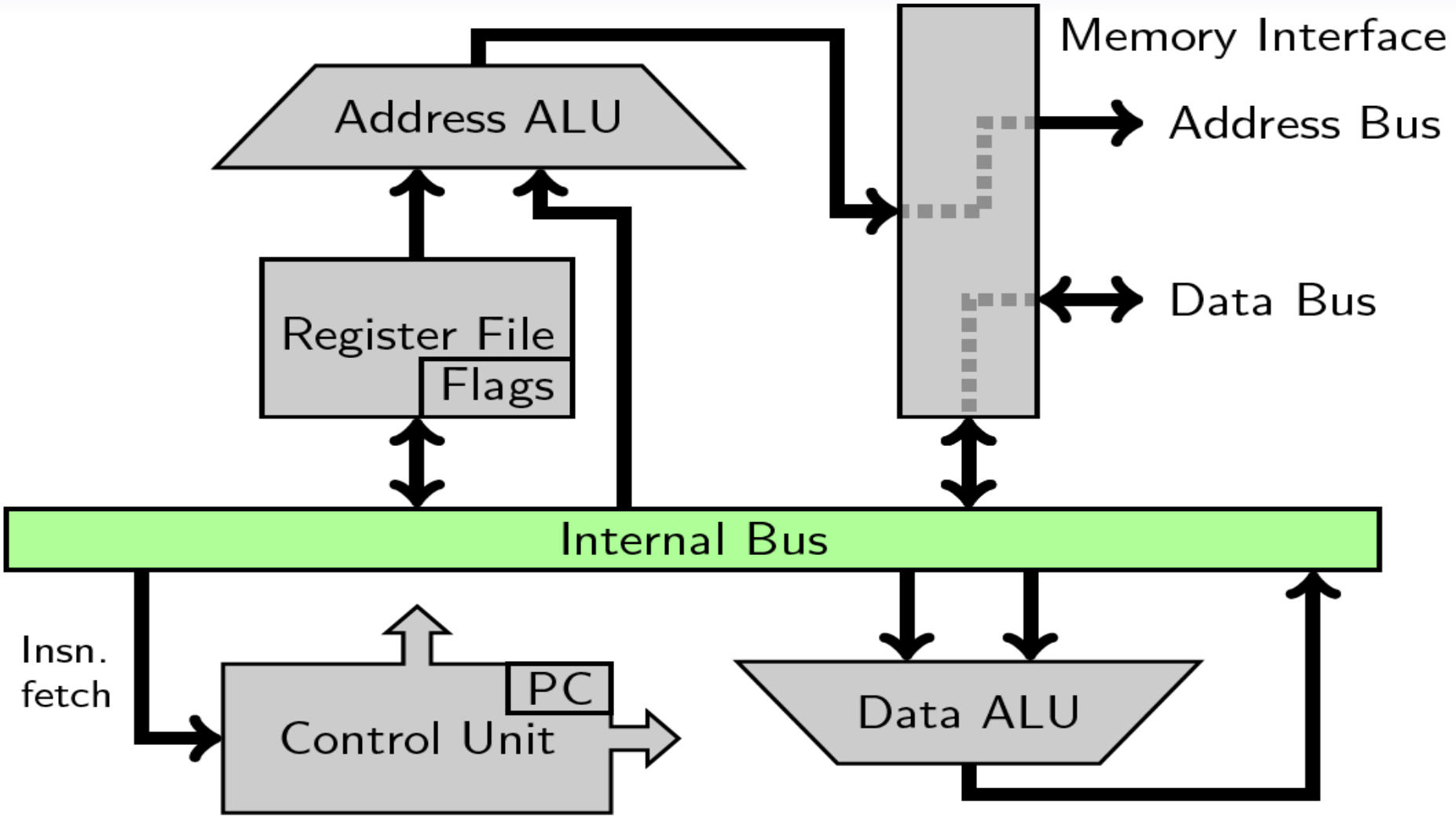
Sprzęt czyli architektury systemów równoległych

Architektura von Neumanna

- Program i dane w pamięci komputera
- Pojedynczy procesor:
  - pobiera rozkaz z pamięci
  - rozkodowuje rozkaz i znajduje adresy argumentów
  - pobiera dane z pamięci
  - wykonuje operacje na danych
  - zapisuje wynik w pamięci

Architektura von Neumanna



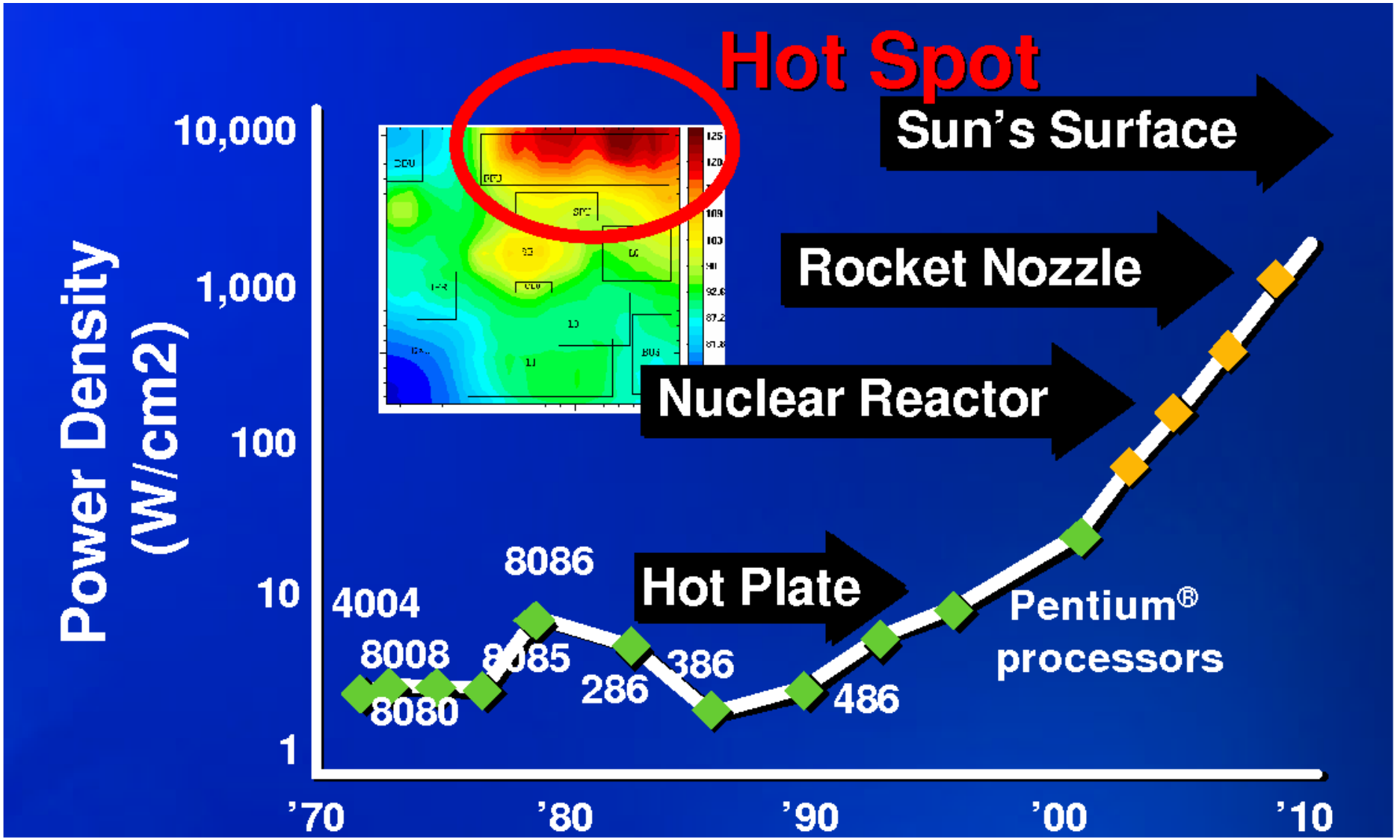
Klasyfikacja Flynna

- Zwielokrotnienie strumieni przetwarzania rozkazów i danych
  - klasyfikacja Flynna
    - SISD (oryginalna architektura von Neumanna)
    - SIMD – jeden strumień rozkazów i wiele strumieni danych
    - MISD – wiele strumieni rozkazów i jeden strumień danych – nie stosowane w praktyce (nie jest to przetwarzanie potokowe, gdzie różne rozkazy są wykonywane na tym samym egzemplarzu danych, ale w kolejnych chwilach czasu)
    - MIMD – wiele strumieni danych i wiele strumieni rozkazów
  - współczesne systemy komputerowe są złożone, realizują zazwyczaj różne typy przetwarzania, z pojedynczymi elementami odpowiadającymi architekturom SISD, SIMD oraz całością odpowiadającą architekturze MIMD – z pamięcią wspólna lub rozproszoną

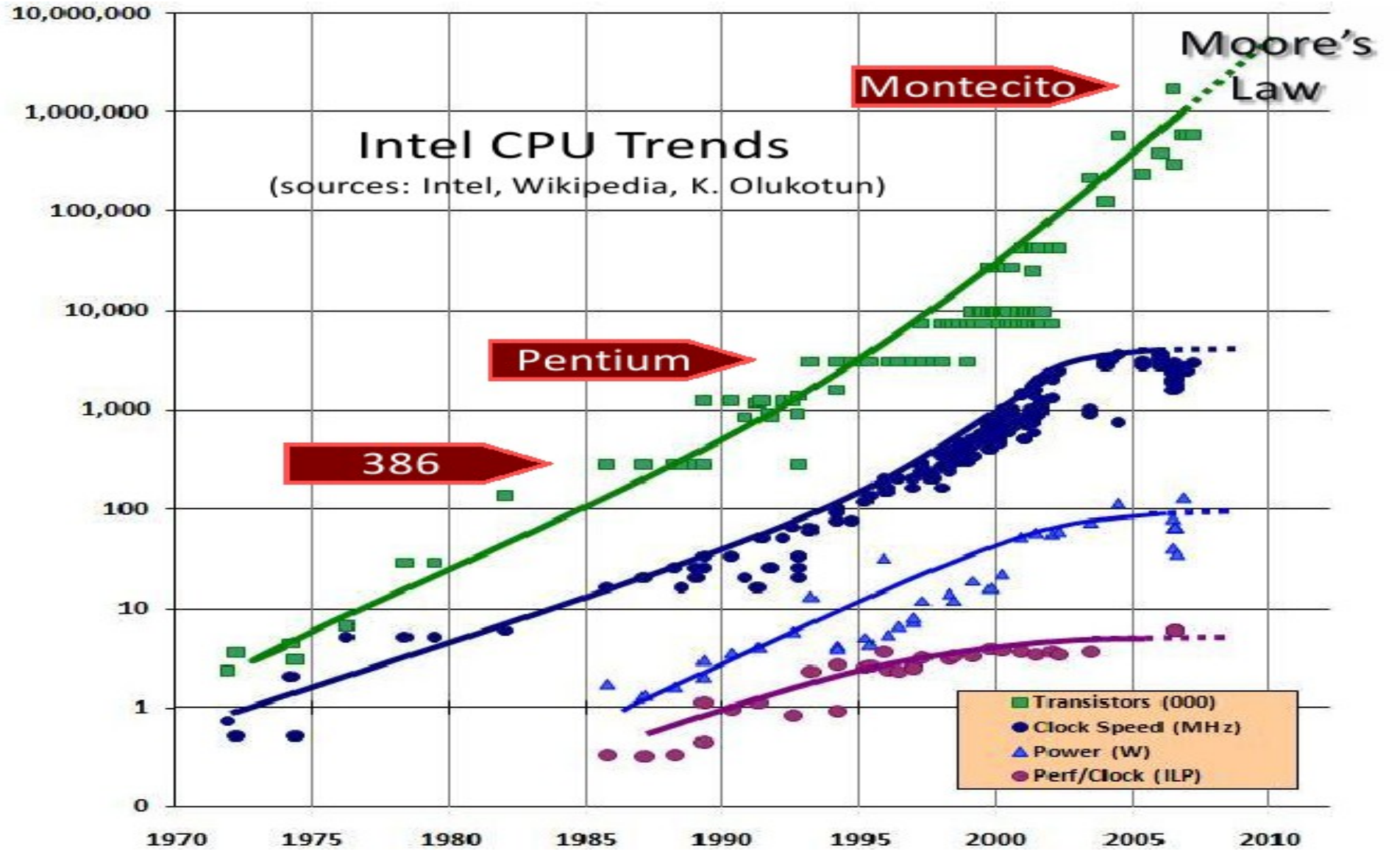
Procesory wielordzeniowe

- Prawo Moore'a – wciąż sprawdzające się w praktyce – mówi o podwajaniu liczby tranzystorów w pojedynczym układzie scalonym co 18 miesięcy
- Wykorzystanie tych możliwości do mnożenia etapów przetwarzania potokowego, liczby jednostek funkcjonalnych pracujących nad jednym strumieniem rozkazów i zwiększania częstości taktowania procesorów – doprowadziło do kryzysu związanego z wydzielaniem ciepła
- Rozwiązaniem tego kryzysu jest wykorzystanie prawa Moore'a do umieszczania w jednym układzie wielu rdzeni (czyli praktycznie niewiele okrojonych procesorów)
  - od początków XXI wieku prawo Moore'a może być tłumaczone jako podwajanie liczby rdzeni w pojedynczym układzie

Wydzielanie ciepła przez procesory



Kierunki rozwoju procesorów Intel



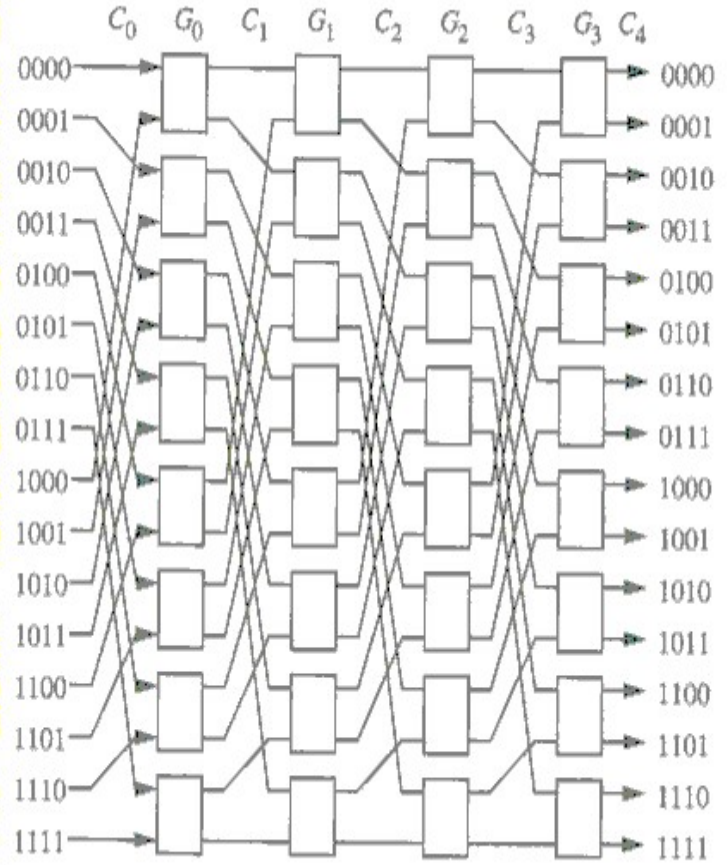
Sieci połączeń w systemach równoległych

- Rodzaje sieci połączeń:
  - podział ze względu na łączone elementy:
    - połączenia procesory-pamięć (moduły pamięci)
    - połączenia międzyprocesorowe (międzywęzłowe)
  - podział ze względu na charakterystyki łączenia:
    - sieci statyczne – zbiór połączeń dwupunktowych
    - sieci dynamiczne – przełączniki o wielu dostępach

Sieci połączeń

- Sieci (połączenia) dynamiczne
  - magistrala *(bus)*
  - krata przełączników (*crossbar switch* - przełącznica krzyżowa)
  - sieć wielostopniowa (*multistage network*)

- Sieć wielostopniowa Ω
  - p wejść i p wyjść, stopnie pośrednie (ile?)
  - każdy stopień pośredni złożony z p/2 przełączników 2x2
  - poszczególne stopnie połączone w sposób realizujący idealne przetasowanie (*perfect shuffle*) (j=2i lub j=2i+1-p)
  - przy dwójkowym zapisie pozycji wejść i wyjść idealne przetasowanie odpowiada rotacji bitów, przełączniki umożliwiają zmianę wartości ostatniego bitu



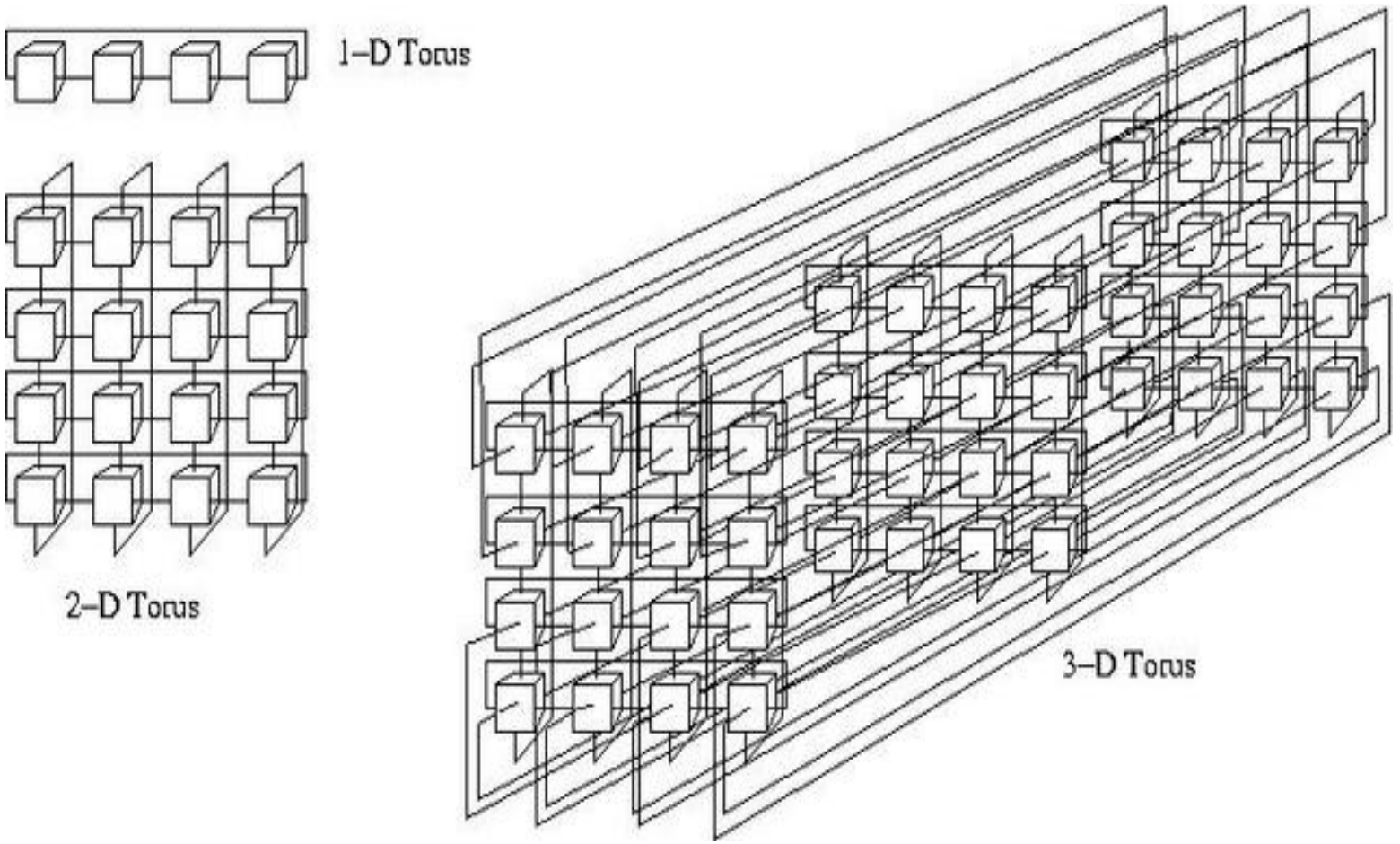
Sieci połączeń

- Porównanie sieci dynamicznych:
  - wydajność
    - szerokość pasma transferu (przepustowość)
    - możliwość blokowania połączeń
  - koszt
    - liczba przełączników
  - skalowalność
    - zależność wydajności i kosztu od liczby procesorów

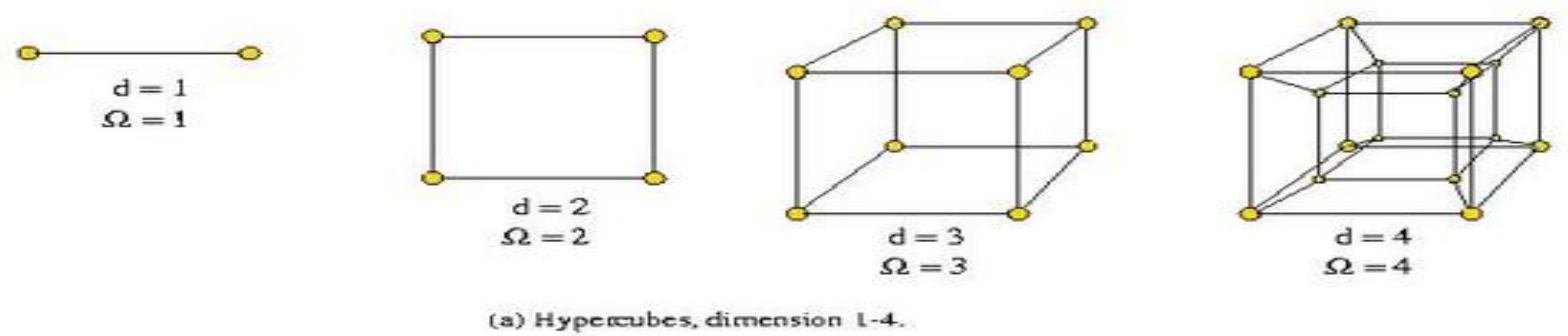
Sieci połączeń

- Sieci statyczne
  - Sieć w pełni połączona
  - Gwiazda
  - Kraty: 1D, 2D, 3D
  - Kraty z zawinięciem, torusy
  - Drzewa: zwykłe lub tłuste
  - Hiperkostki: 1D, 2D, 3D itd..
    - Wymiar – d, liczba procesorów – 2^d
    - Bitowy zapis położenia węzła
    - Najkrótsza droga między 2 procesorami = ilość bitów, którymi różnią się kody położenia procesorów

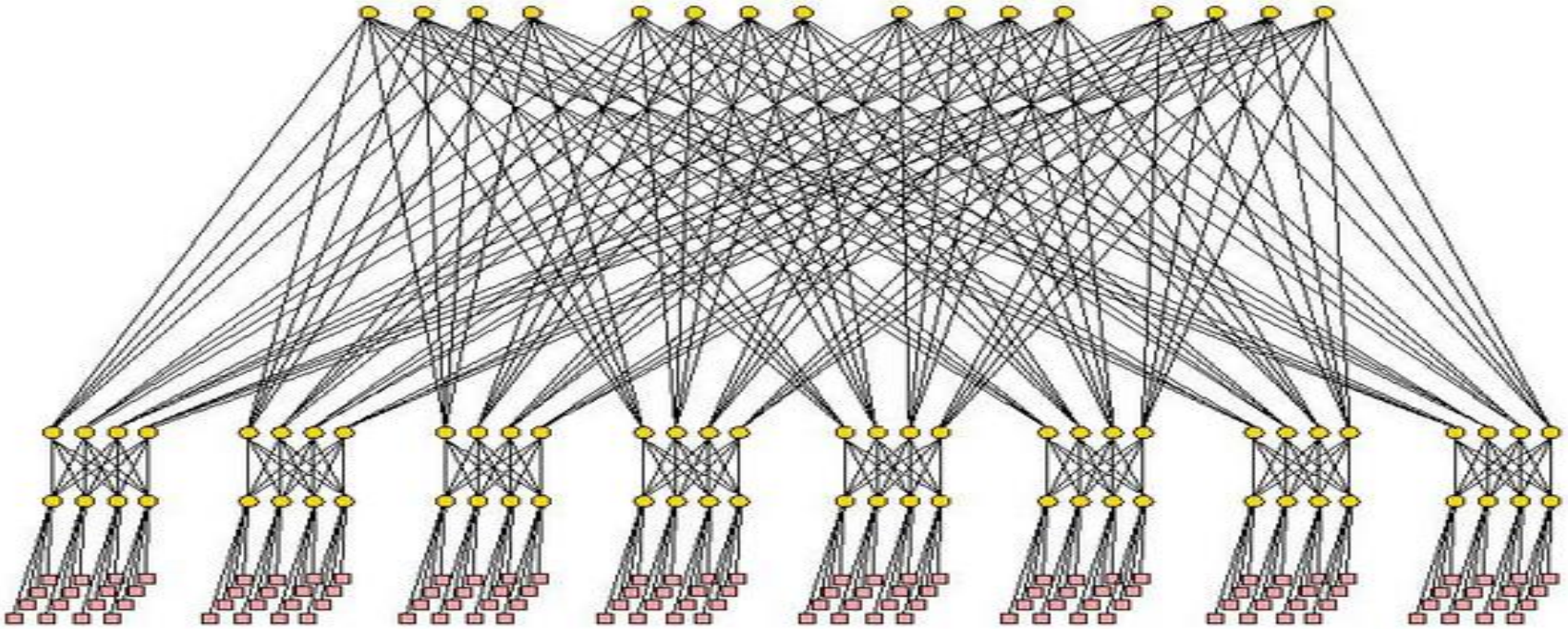
Topologia torusa



Topologie hiperkostki i drzewa

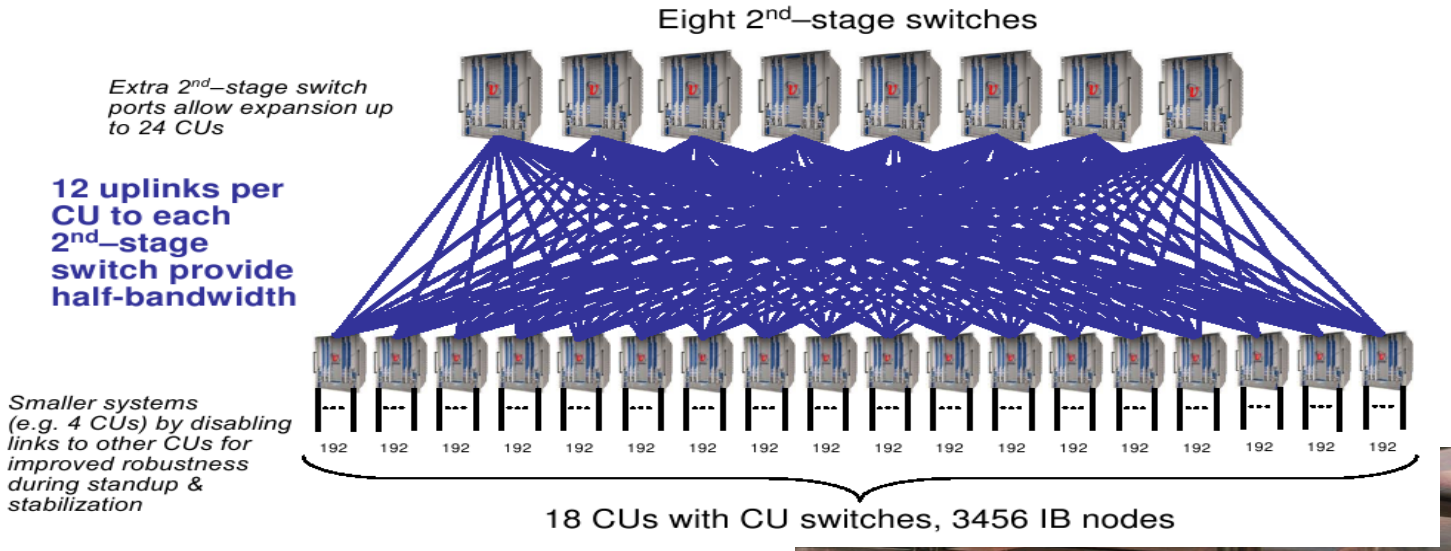


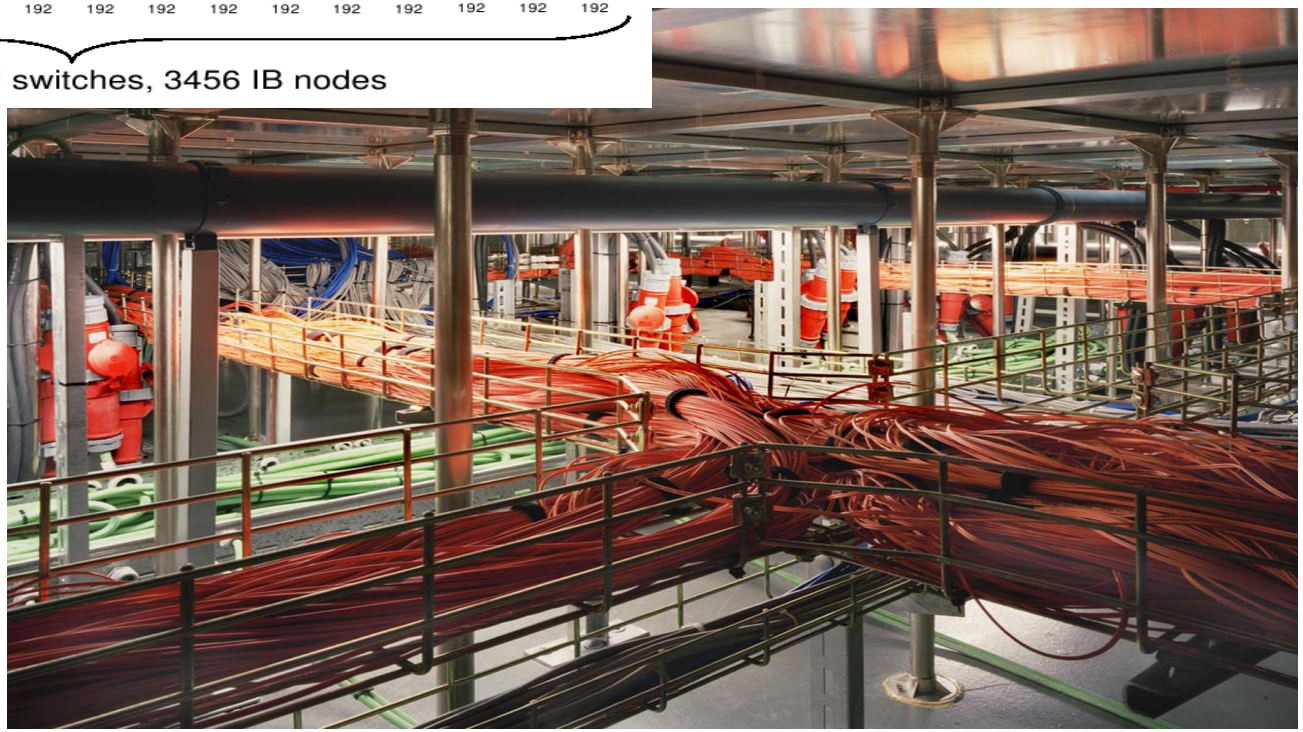
(a) Hypercubes, dimension 1-4.



(b) A 128-way fat tree.

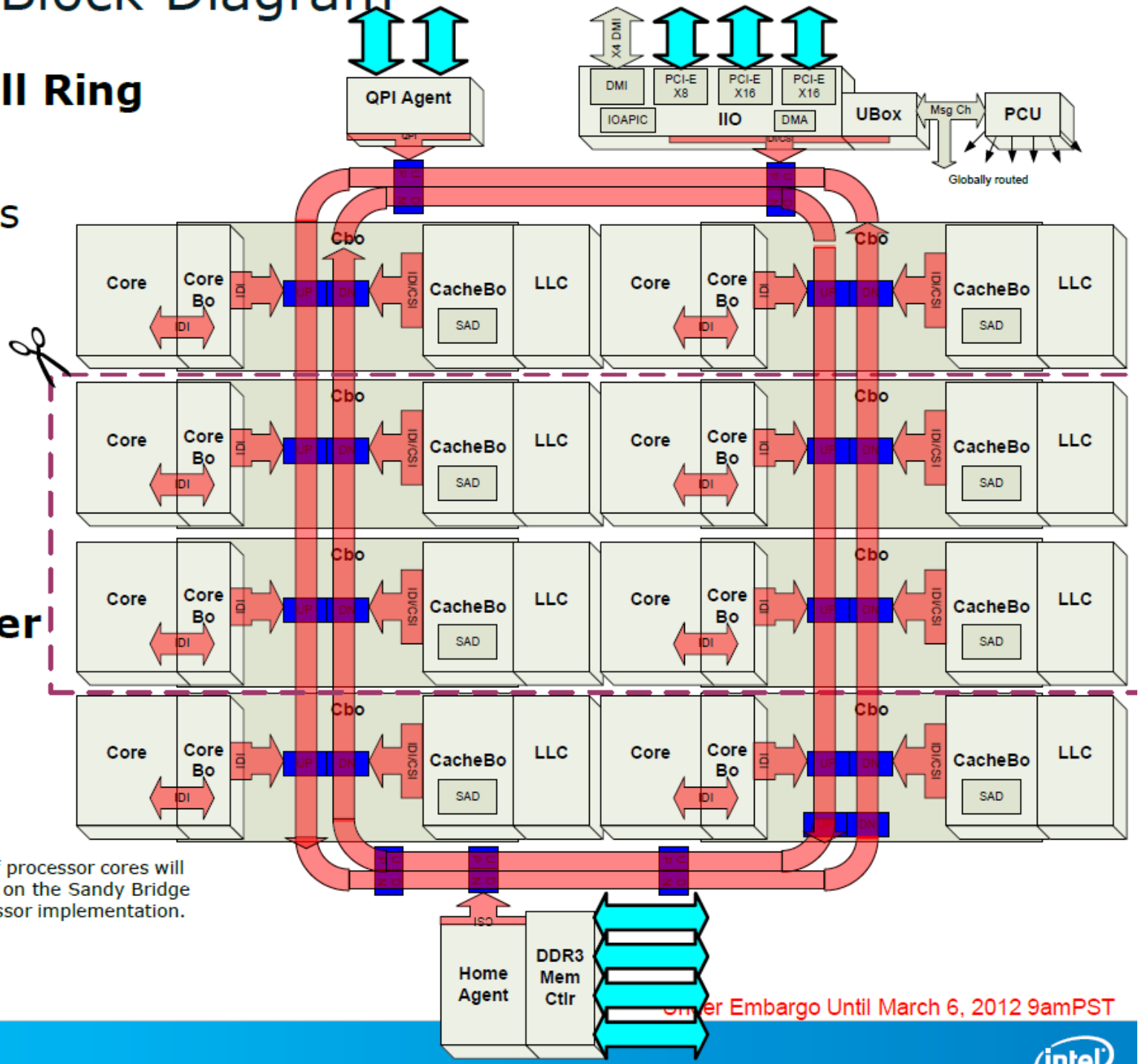
Sieci połączeń





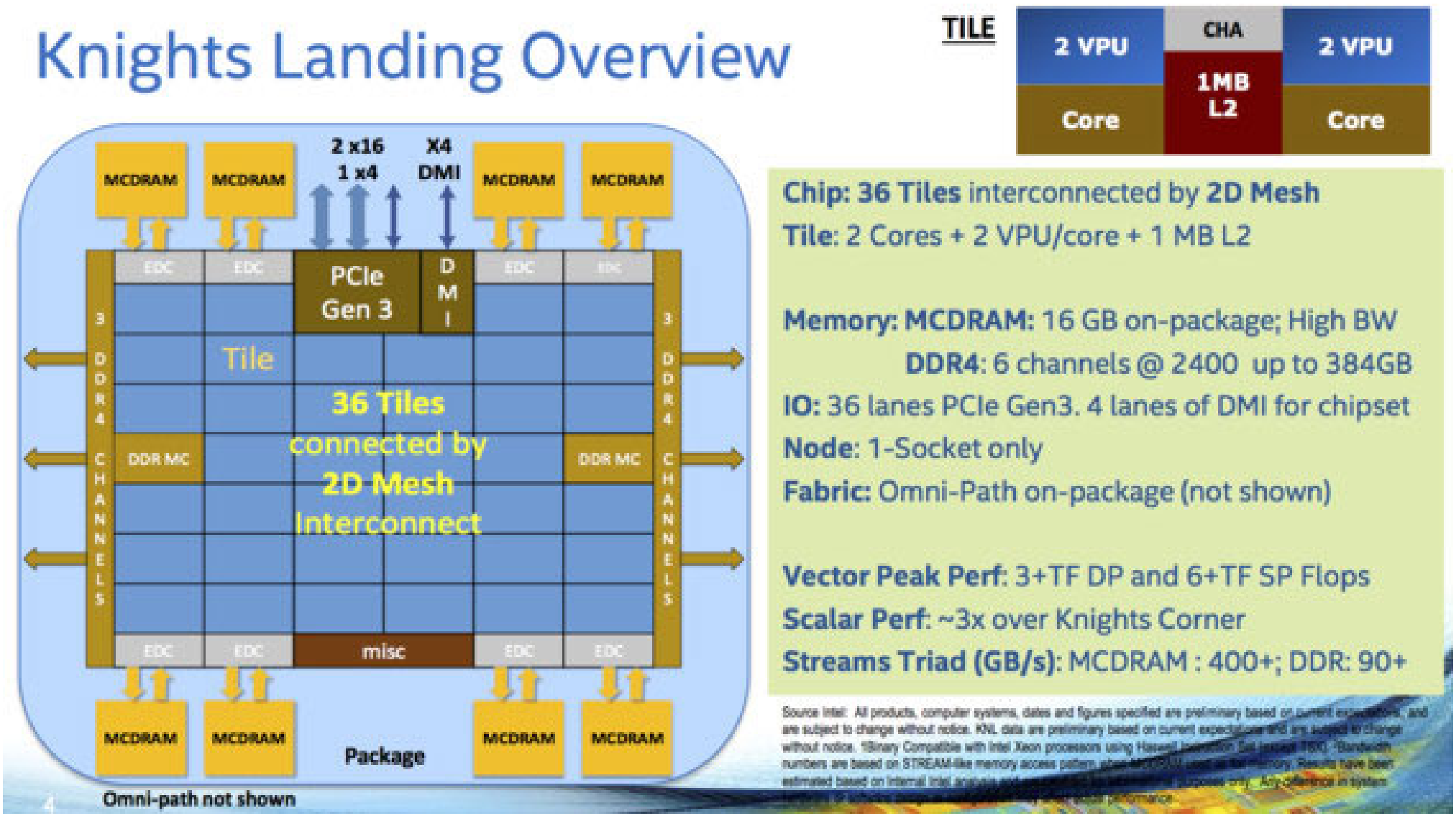
Xeon E5-2600 Block Diagram

- **Bi-Directional Full Ring**
  - 32B/clk/agent
  - 8 Core/LLC slices
- **Last Level Cache**
  - 32B/clk/slice
- **Dual QPI Agent**
- **Integrated I/O**
- **Home Agent**
- **Integerated Memory Controller**
  - Connected to HA
- **PCU**
- **"Ubox"**

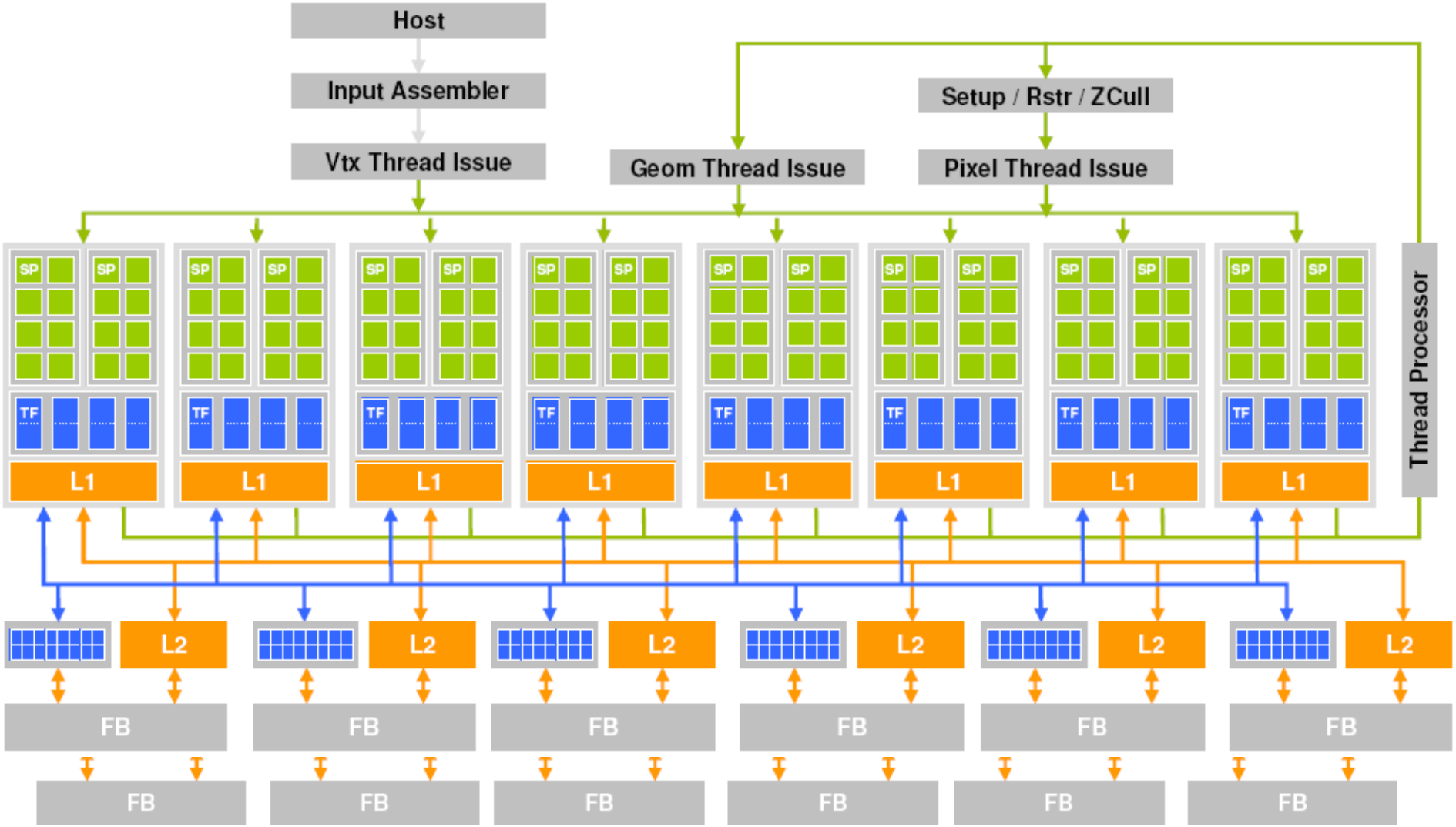


Block Diagram Illustrative only. Number of processor cores will vary with different processor models based on the Sandy Bridge Microarchitecture. Represents server processor implementation.

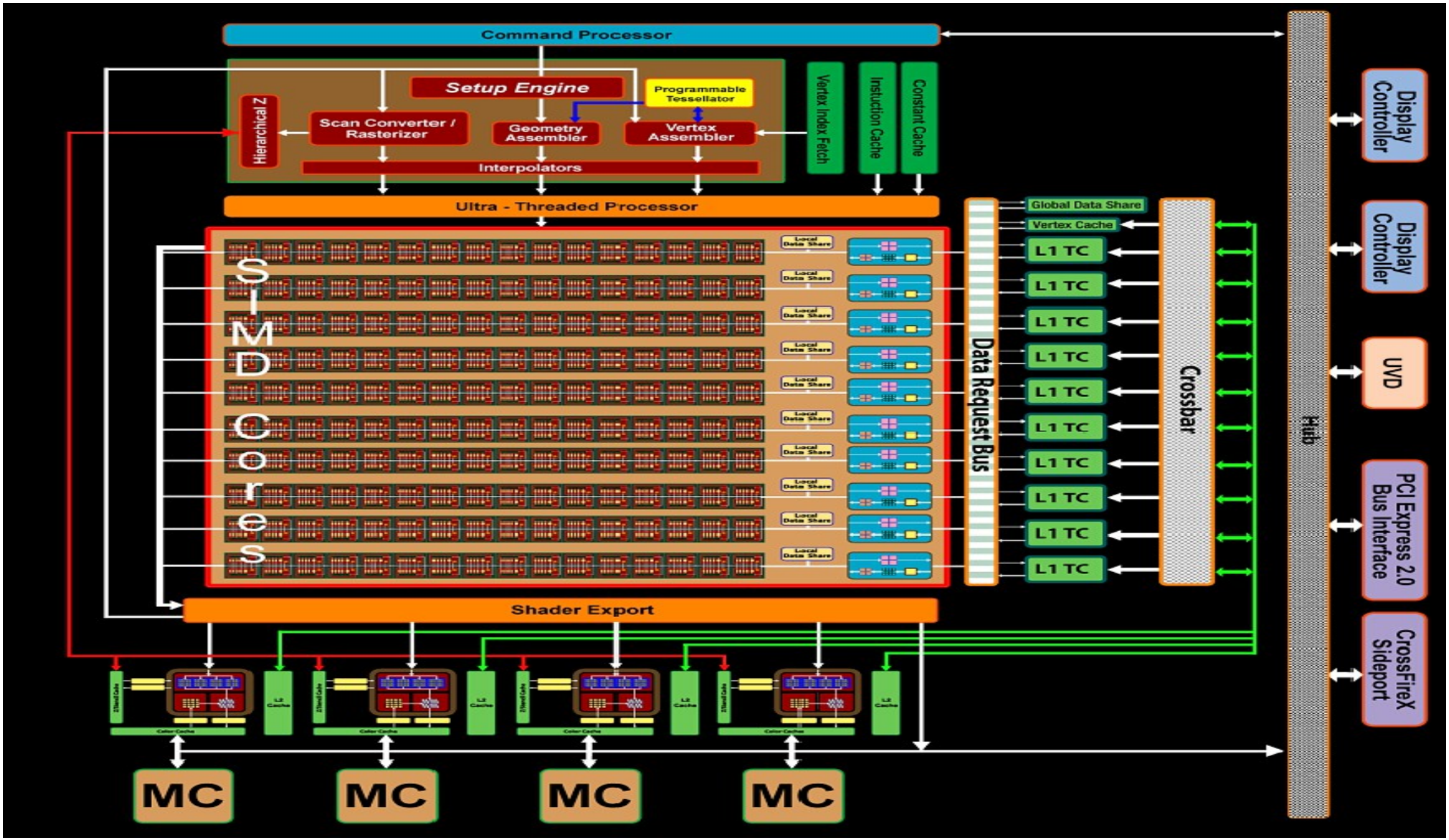
Intel Knights Landing

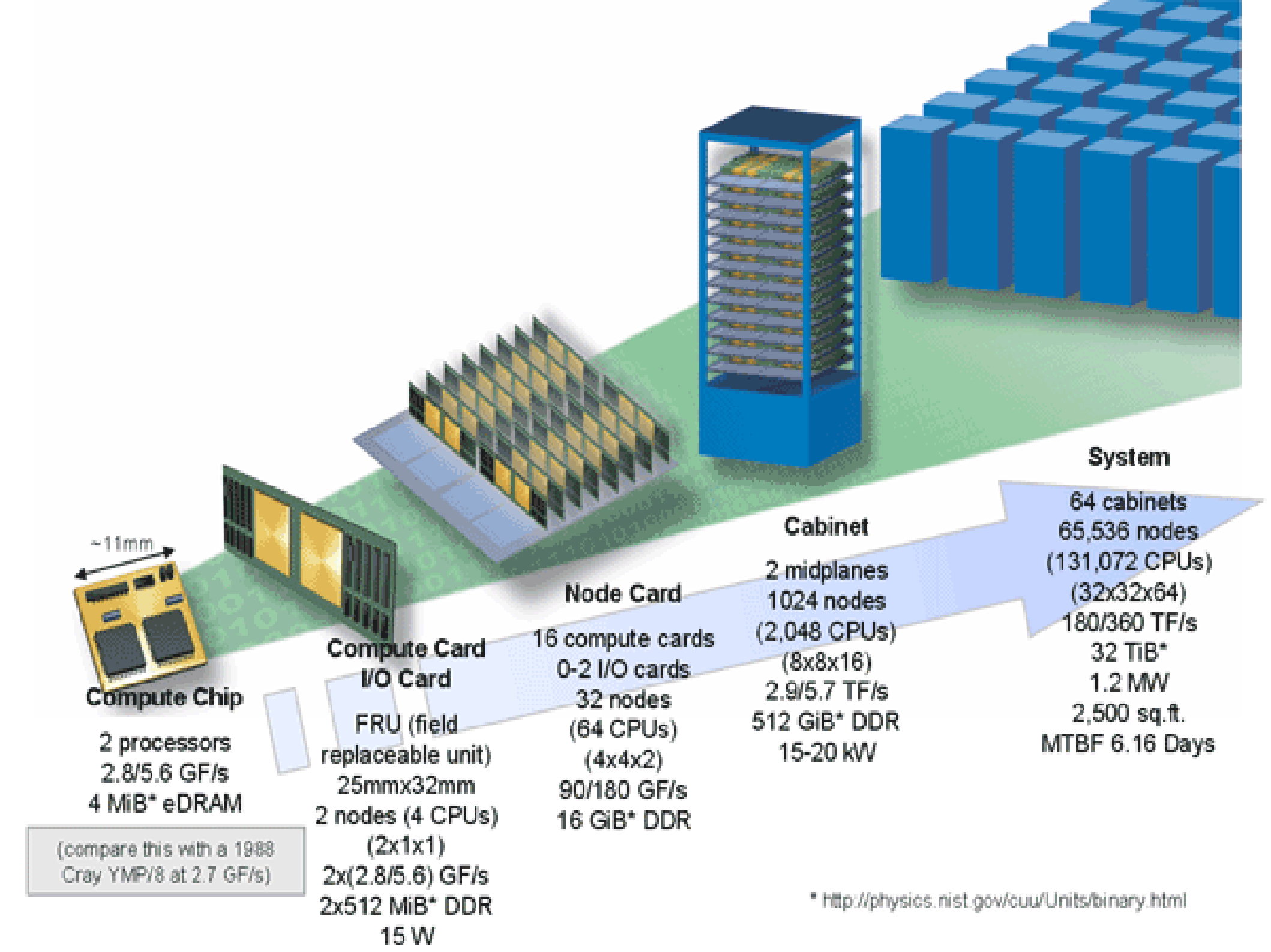


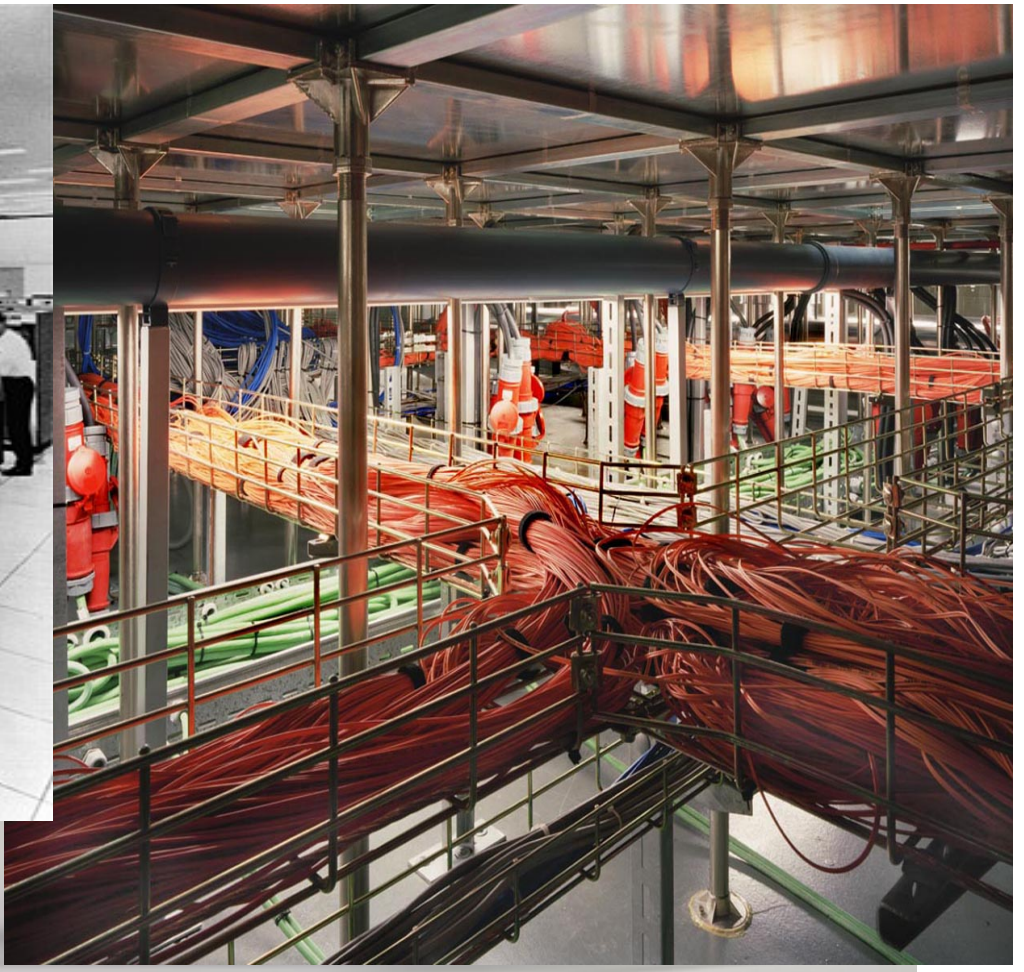
Architektura procesora G80

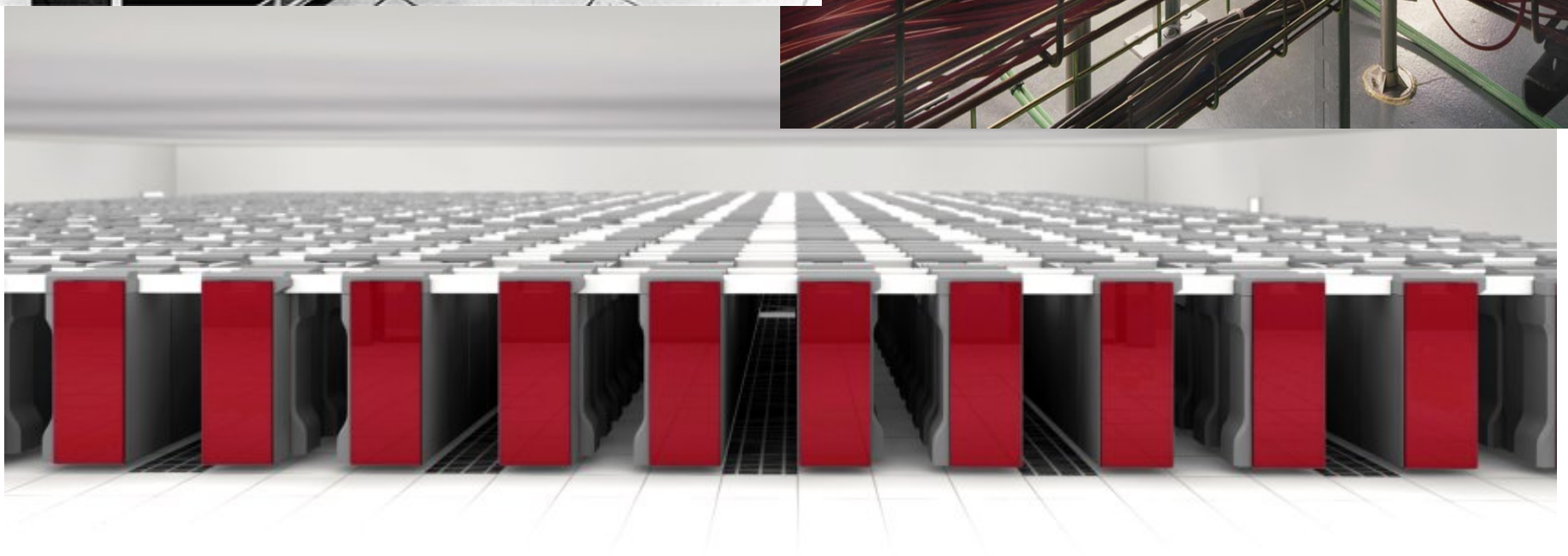


ATI FireStream




~11mm
Compute Chip
2 processors
2.8/5.6 GF/s
4 MiB* eDRAM
(compare this with a 1988 Cray YMP/8 at 2.7 GF/s)
Compute Card
I/O Card
FRU (field replaceable unit)
25mmx32mm
2 nodes (4 CPUs)
(2x1x1)
2x(2.8/5.6) GF/s
2x512 MiB* DDR
15 W
Node Card
16 compute cards
0-2 I/O cards
32 nodes
(64 CPUs)
(4x4x2)
90/180 GF/s
16 GiB* DDR
Cabinet
2 midplanes
1024 nodes
(2,048 CPUs)
(8x8x16)
2.9/5.7 TF/s
512 GiB* DDR
15-20 kW
System
64 cabinets
65,536 nodes
(131,072 CPUs)
(32x32x64)
180/360 TF/s
32 TiB*
1.2 MW
2,500 sq.ft.
MTBF 6.16 Days
* http://physics.nist.gov/cuu/Units/binary.html





Alternatywne modele programowania równoległego

PRAM

- Teoretyczne modele obliczeń (do analizy algorytmów)
  - maszyna o dostępie swobodnym (RAM)
    - procesor, rejestry, magistrala, pamięć
  - równoległa maszyna o dostępie swobodnym (PRAM)
    - wiele procesorów (z własnymi rejestrami), magistrala, wspólna pamięć
    - równoległa realizacja algorytmów poprzez odpowiednie numerowanie procesorów i powiązanie numeracji z realizacją operacji
    - założona automatyczna (sprzętowa) synchronizacja działania procesorów – pominięcie zagadnień czasowej złożoności synchronizacji i komunikacji (zapisu i odczytu z i do pamięci)

PRAM

- Teoretyczne modele obliczeń (do analizy algorytmów)
  - równoległa maszyna o dostępie swobodnym (PRAM)
    - EREW – wyłączny odczyt, wyłączny zapis
    - CREW– jednoczesny odczyt, wyłączny zapis
    - ERCW– wyłączny odczyt, jednoczesny zapis
    - CRCW– jednoczesny odczyt, jednoczesny zapis; strategie:
      - jednolita (*common)* – dopuszcza zapis tylko identycznych danych
      - dowolna (*arbitrary)* – wybiera losowo dane do zapisu
      - *priorytetowa (priority)* – wybiera dane na zasadzie priorytetów

Algorytmy równoległe dla modelu PRAM

- Algorytm obliczania elementu maksymalnego tablicy A o rozmiarze *n* na n^2 procesorowej maszynie CRCW PRAM ze strategią jednolitą:

```
element_maks( A, n ): maks{ // uchwyt do tablicy i skrajne indeksy
    DLA i= 1,...,n { || m[i] := PRAWDA } // tablica pomocnicza m
    DLA i=1,...,n{              // dzięki użyciu n^2 procesorów równoległe
        DLA j=1,...,n{          // wykonanie pętli zajmuje O(1) czasu
    ||  JEŻELI( A[i] < A[j] ) m[i] :=   FAŁSZ
    } }
    DLA i=1,...,n { ||  JEŻELI( m[i]=PRAWDA ) maks := A[i] }
    ZWRÓĆ maks
}
```

Model równoległości danych

- Model SPMD (pierwotnie dla maszyn SIMD)
- Zrównoleglenie poprzez przypisanie danych poszczególnym procesorom
- Wymiana informacji pomiędzy procesorami bez jawnego sterowania przez programistę
- Implementacja powinna umożliwiać efektywną realizację programów, tak dla systemów z pamięcią wspólną, jak i dla środowisk przesyłania komunikatów bez pamięci wspólnej
- Realizacja powyższych ambitnych celów jest na tyle trudna, że model nie doczekał się jeszcze rozpowszechnionych implementacji ogólnego przeznaczenia

High Performance Fortran

- Specyfikacja – nieformalny standard
- Rozszerzenie Fortranu 95
- Realizacja modelu programowania z równoległością danych, głównie dla operacji wektorowych i macierzowych
- Rozszerzenia w postaci dyrektyw kompilatora uzupełniających standardowe instrukcje Fortranu
- Z założenia ma udostępniać model programowania prostszy niż przesyłanie komunikatów i równie efektywny

Fortran 95

- Specyfika Fortranu 95 wiąże się z rozbudowanymi operacjami na wektorach i macierzach (tablicach wielowymiarowych):
  - deklaracje dopuszczają macierze wielowymiarowe:

    ```
    REAL, DIMENSION(3,4):: A, B, C
    ```

  - zakres poszczególnych indeksów określa kształt macierzy
  - istnieją zdefiniowane operacje dotyczące wektorów i macierzy (iloczyn, suma, itp) oraz pętle operujące na wszystkich lub wybranych elementach wektorów lub macierzy

Fortran 95

- cd. operacje na wektorach i macierzach:
  - dopuszczalne jest odnoszenie się do wybranych fragmentów wektorów i macierzy (każdy fragment ma swój kształt i może być użyty w odpowiednich operacjach), np.:

    ```
    AJ(1:3, 1:3) = OUGH(4, 3:5, 4:8:2)
    ```

  - w pętlach wektorowych można stosować maskowanie za pomocą operacji logicznych lub wektorów o wartościach logicznych:

    ```
    WHERE( AJ.NE.0 )  A(1:3) = 1/A(1:3)
    ```

  - istnieje wiele wektorowych procedur bibliotecznych, np. SUM, PRODUCT, MINVAL, MAXVAL, itp.

High Performance Fortran

- Dyrektywy określające ułożenie procesorów:

```
!HPF$    PROCESSORS, DIMENSION(3,4):: P1
!HPF$    PROCESSORS, DIMENSION(2,6):: P2
!HPF$    PROCESSORS, DIMENSION(12):: P3
!HPF$    PROCESSORS, DIMENSION(2,2,3):: P4
```

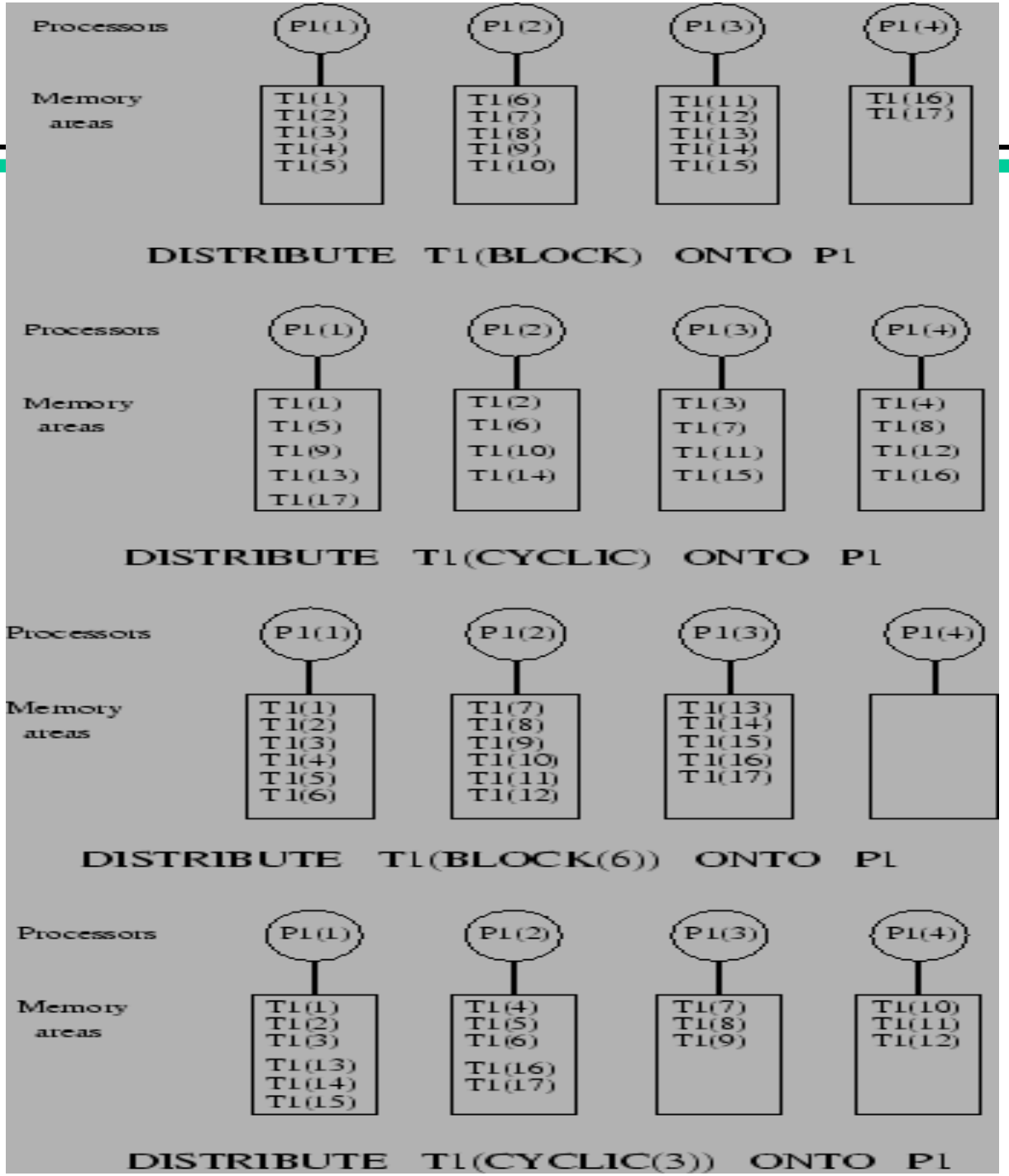
  - W jednym programie może istnieć wiele zdefiniowanych układów procesorów, wykorzystywanych w różnych miejscach do realizacji różnych operacji

High Performance Fortran

- Przypisanie (dystrybucja) danych odbywa się za pomocą dyrektywy DISTRIBUTE, np.:

```
!HPF$ PROCESSORS, DIMENSION(4):: P
REAL, DIMENSION(17):: T1, T2
!HPF$ DISTRIBUTE   T1(BLOCK)   ONTO   P
!HPF$ DISTRIBUTE   T1(CYCLIC)   ONTO   P
!HPF$ DISTRIBUTE   T1( CYCLIC(3) )   ONTO   P
```

  - określenie BLOCK (lub BLOCK(n)) oznacza podział tablicy na bloki przypisywane kolejnym procesorom (n nie może być za małe)
  - określenie CYCLIC (lub CYCLIC(n)) oznacza przypisywanie kolejnych wyrazów tablicy (kolejnych n-tek wyrazów tablicy) kolejnym procesorom z okresowym zawijaniem


Processors
P1(1)
P1(2)
P1(3)
P1(4)
Memory areas
T1(1) T1(2) T1(3) T1(4) T1(5)
T1(6) T1(7) T1(8) T1(9) T1(10)
T1(11) T1(12) T1(13) T1(14) T1(15)
T1(16) T1(17)
DISTRIBUTE T1(BLOCK) ONTO P1
Processors
P1(1)
P1(2)
P1(3)
P1(4)
Memory areas
T1(1) T1(5) T1(9) T1(13) T1(17)
T1(2) T1(6) T1(10) T1(14)
T1(3) T1(7) T1(11) T1(15)
T1(4) T1(8) T1(12) T1(16)
DISTRIBUTE T1(CYCLIC) ONTO P1
Processors
P1(1)
P1(2)
P1(3)
P1(4)
Memory areas
T1(1) T1(2) T1(3) T1(4) T1(5) T1(6)
T1(7) T1(8) T1(9) T1(10) T1(11) T1(12)
T1(13) T1(14) T1(15) T1(16) T1(17)
DISTRIBUTE T1(BLOCK(6)) ONTO P1
Processors
P1(1)
P1(2)
P1(3)
P1(4)
Memory areas
T1(1) T1(2) T1(3) T1(13) T1(14) T1(15)
T1(4) T1(5) T1(6) T1(16) T1(17)
T1(7) T1(8) T1(9)
T1(10) T1(11) T1(12)
DISTRIBUTE T1(CYCLIC(3)) ONTO P1

High Performance Fortran

- Dystrybucja tablic wielowymiarowych odbywa się odrębnie dla każdego wymiaru
- Liczba wymiarów podlegających dystrybucji musi być równa liczbie wymiarów układu procesorów

```
!HPF$    PROCESSORS, DIMENSION(2,2):: P1
!HPF$    PROCESSORS, DIMENSION(4):: P2
REAL, DIMENSION(8,8):: T1
!HPF$    DISTRIBUTE   T1(BLOCK,BLOCK)   ONTO   P1
!HPF$    DISTRIBUTE   T1(CYCLIC,CYCLIC)   ONTO   P1
!HPF$    DISTRIBUTE   T1(BLOCK,CYCLIC)   ONTO   P1
!HPF$    DISTRIBUTE   T1(*,CYCLIC)   ONTO   P2
```

High Performance Fortran

- Dla uzyskania wysokiej wydajności obliczeń (minimalizacji komunikacji międzyprocesorowej) dystrybucja niektórych tablic jest przeprowadzana zgodnie z dystrybucją innych przez wykorzystanie dyrektywy ALIGN:

```
!HPF$ ALIGN A(:) WITH B(:)
!HPF$ ALIGN C(i,j) WITH D(j,i)
!HPF$ ALIGN C(*,j) WITH A(j)  // zwijanie wymiaru
!HPF$ ALIGN B(j) WITH D(j,*)  // powielanie wyrazów
```

- Wyrównanie może być zdefiniowane dla pustego wzorca (*template)* zadeklarowanego tak jak rzeczywiste tablice

```
!HPF$   TEMPLATE   T(4,5)
```

High Performance Fortran

- Wykonanie równoległe odbywa się poprzez realizację:
  - standardowych operacji wektorowych Fortranu 95
    - A+B, A*B, A-(B*C) – operacje dozwolone dla macierzy zgodnych *(conformable)*
  - pętli operujących na wszystkich elementach macierzy:

```
FORALL ( i=1..n, j=1..m, A[i,j].NE.0 )   A(i,j) = 1/A(i,j)
```

- Równoległość jest uzyskiwana niejawnie – programista nie określa, które operacje są wykonywane na konkretnym procesorze

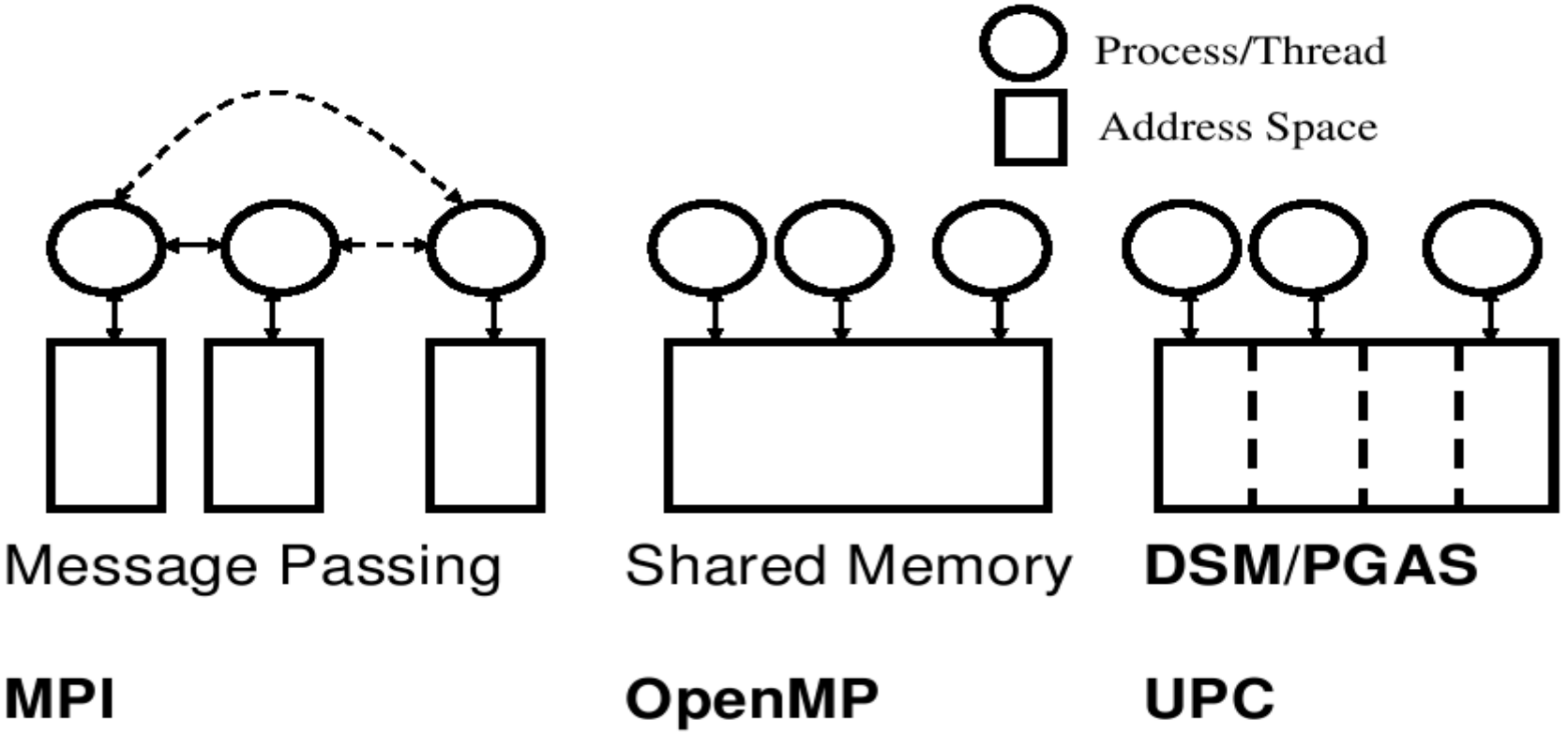
High Performance Fortran

- Przykład: dekompozycja LU macierzy A

```
      INTEGER M, N, R, R1

!HPF$ PROCESSORS P (M, M)

!HPF$ TEMPLATE T (N, N)
!HPF$ DISTRIBUTE T (CYCLIC, CYCLIC) ONTO P

      REAL A (N, N)
!HPF$ ALIGN A (I, J) WITH T (I, J)

      REAL, DIMENSION (SIZE (A, 1)) :: L_COL, U_ROW
!HPF$ ALIGN L_COL (I) WITH T (I, *)
!HPF$ ALIGN U_ROW (J) WITH T (*, J)

      DO R = 1, N - 1
        R1 = R + 1
        L_COL (R  : ) = A (R : , R)
        A (R , R1 : ) = A (R, R1 : ) / L_COL (R)
        U_ROW (R1 : ) = A (R, R1 : )
        FORALL (I = R1 : N, J = R1 : N)
     &    A (I, J) = A (I, J) - L_COL (I) * U_ROW (J)
      ENDDO
```

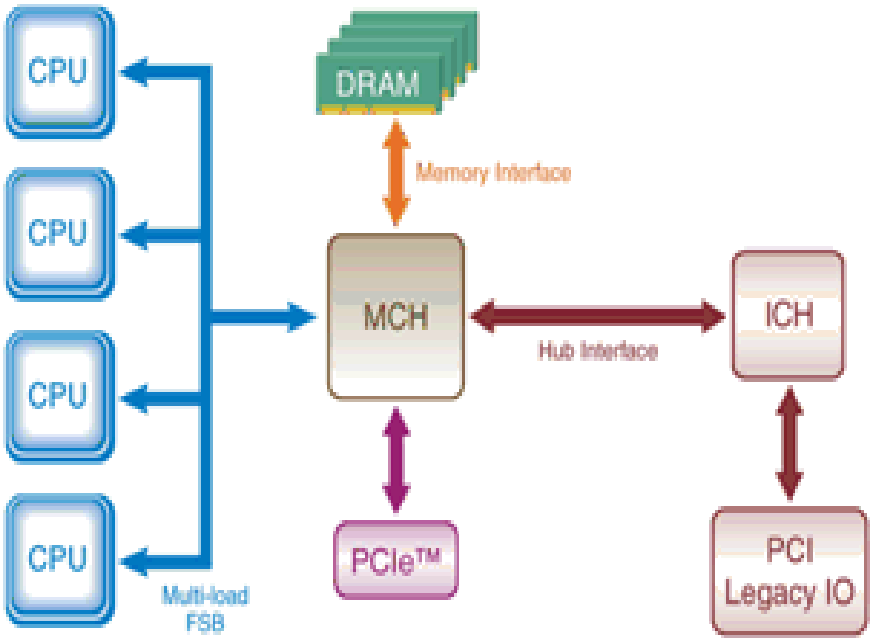
Model DSM/PGAS

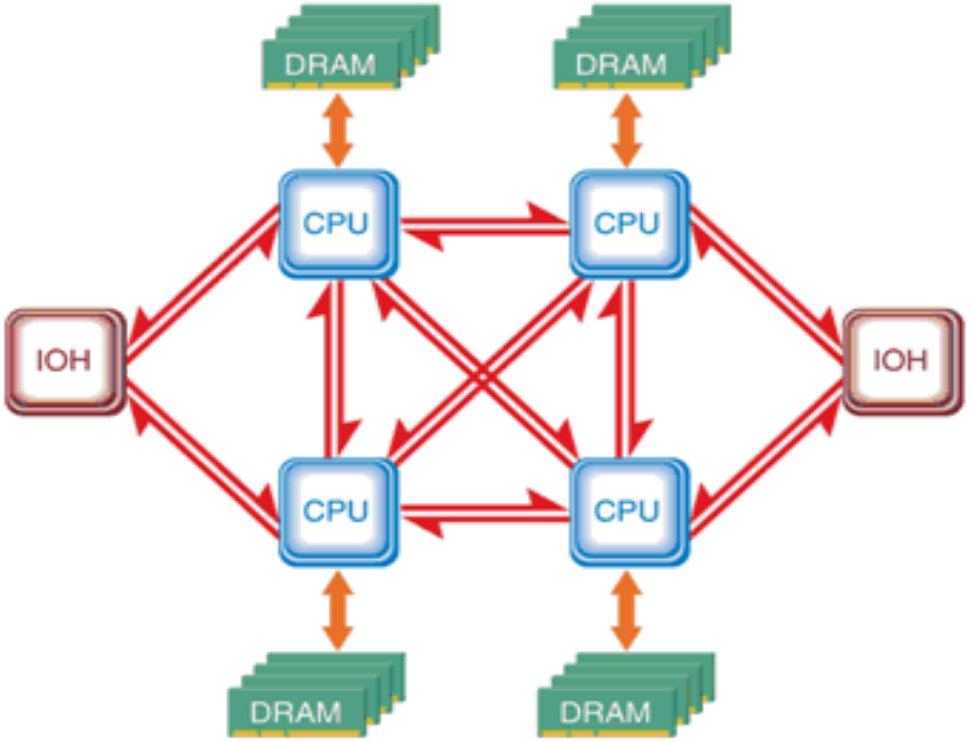
- Model programowania z rozproszoną pamięcią wspólną (DSM – *distributed shared memory) (*inna nazwa – PGAS, *partitioned global adress space*, dzielona globalna przestrzeń adresowa)



SMP, UMA, NUMA, etc.

- UMA – SMP





- NUMA – DSM
- ccNUMA

Model DSM/PGAS

- Idee rozwijane od wielu lat w językach takich jak
  - CoArray Fortran
  - Unified Parallel C
  - inne
- Przystosowane do maszyn NUMA
  - jawna kontrola nad przyporządkowaniem danych do wątków (pojęcie afiniczności)
  - możliwość optymalizacji
  - trudność synchronizacji i utrzymania zgodności przy dostępach do pamięci wspólnej
- Wzrost zainteresowania w momencie popularyzacji systemów wieloprocesorowych i, w dalszej perspektywie, procesorów wielordzeniowych o architekturze NUMA

Unified Parallel C

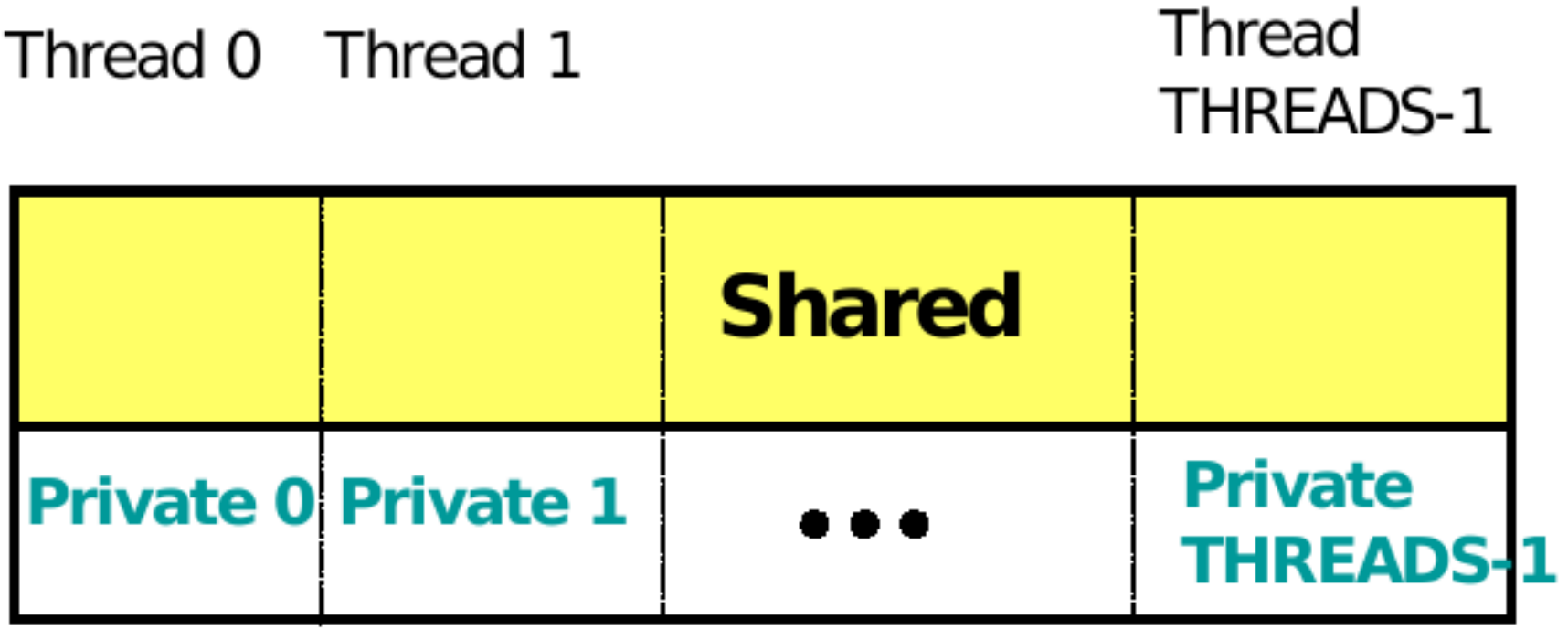
- UPC (*Unified Parallel C*)
  - rozszerzenie języka C
    - nowe słowa kluczowe
    - zestaw standardowych procedur bibliotecznych
      - m.in. operacje komunikacji grupowej
  - model programowania DSM/PGAS (*distributed shared memory, partitioned global adress space)*
  - model programowania SPMD
  - specyfikacja jest modyfikowana (1.0 2001, 1.1 2003, 1.2 2005)
  - szereg implementacji (Cray, HP, UCB, GWU, MTU i inne)
  - środowiska tworzenia kodu - „debuggery”, analizatory, itp.

Unified Parallel C

- Sterowanie wykonaniem:
  - THREADS – liczba wątków
  - MYTHREAD – identyfikator (ranga) wątku
  - klasyczny model (podobnie jak w MPI)
  - polecenie upc_forall (podobnie jak pętla równoległa w OpenMP)
  - określenie liczby wątków w trakcie kompilacji (nowość!) lub w momencie uruchamiania programu
  - bariery (w postaci procedur – blokujących i nieblokujących)
  - zamki (podobnie jak mutex-y)
  - synchronizacja związana z dostępem do zmiennych wspólnych (konieczność utrzymania zgodności)

Unified Parallel C

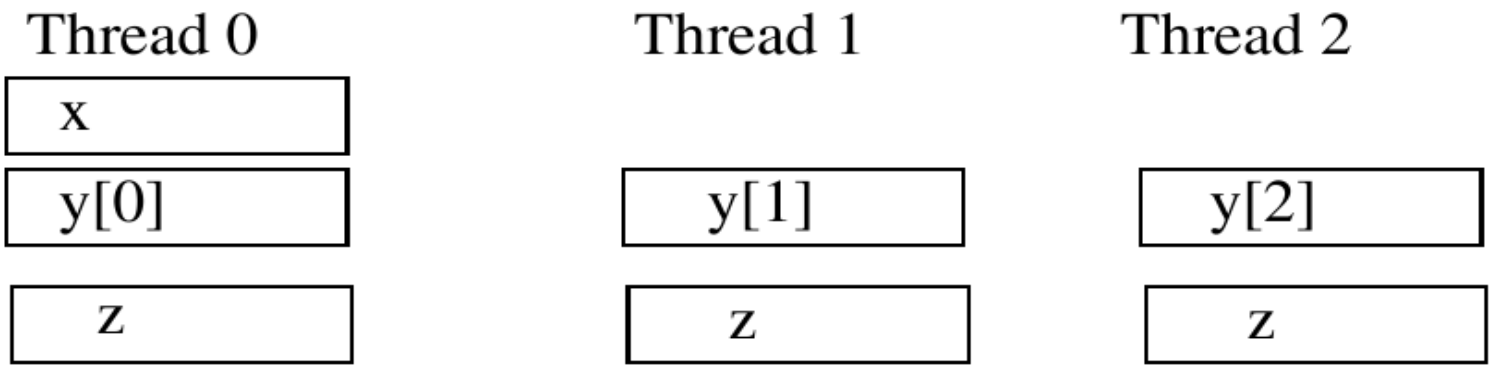
- Zarządzanie pamięcią (dostęp do zmiennych):
  - zmienne lokalne – dostępne wątkowi właścicielowi
  - zmienne globalne – dostępne wszystkim, afiniczność



UPC

- Słowo kluczowe „shared” - deklaracja umieszczenia zmiennej lub tablicy w pamięci wspólnej – afiniczność każdego elementu do pojedynczego wątku

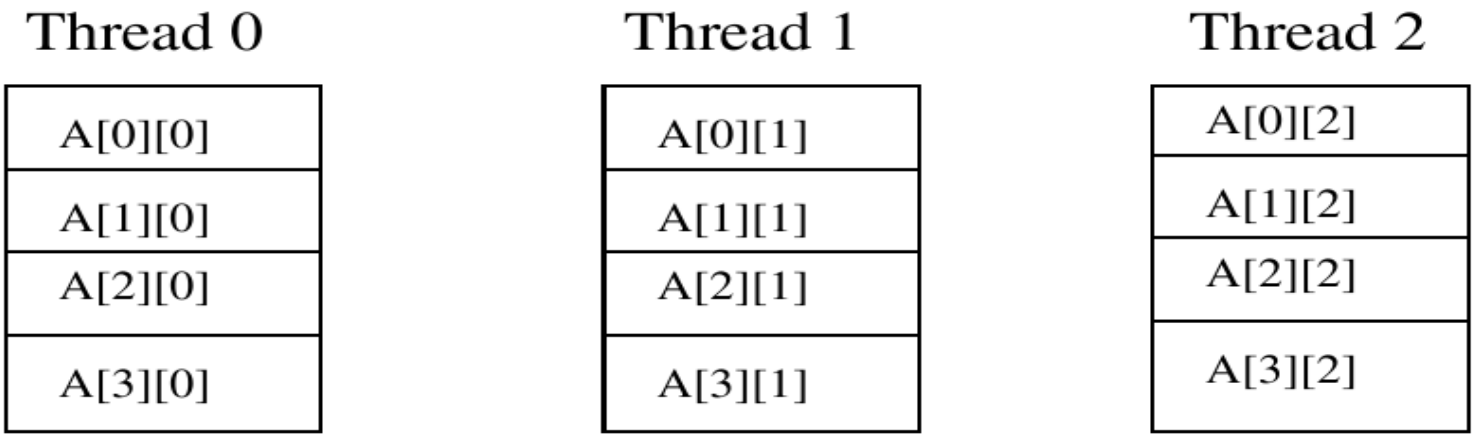
```
shared int x;  /*x will have affinity to thread 0 */
shared int y[THREADS];
int z;
```



UPC

- Rozkład tablic wielowymiarowych oparty jest na standardowym dla C rozłożeniu elementów (wierszami)

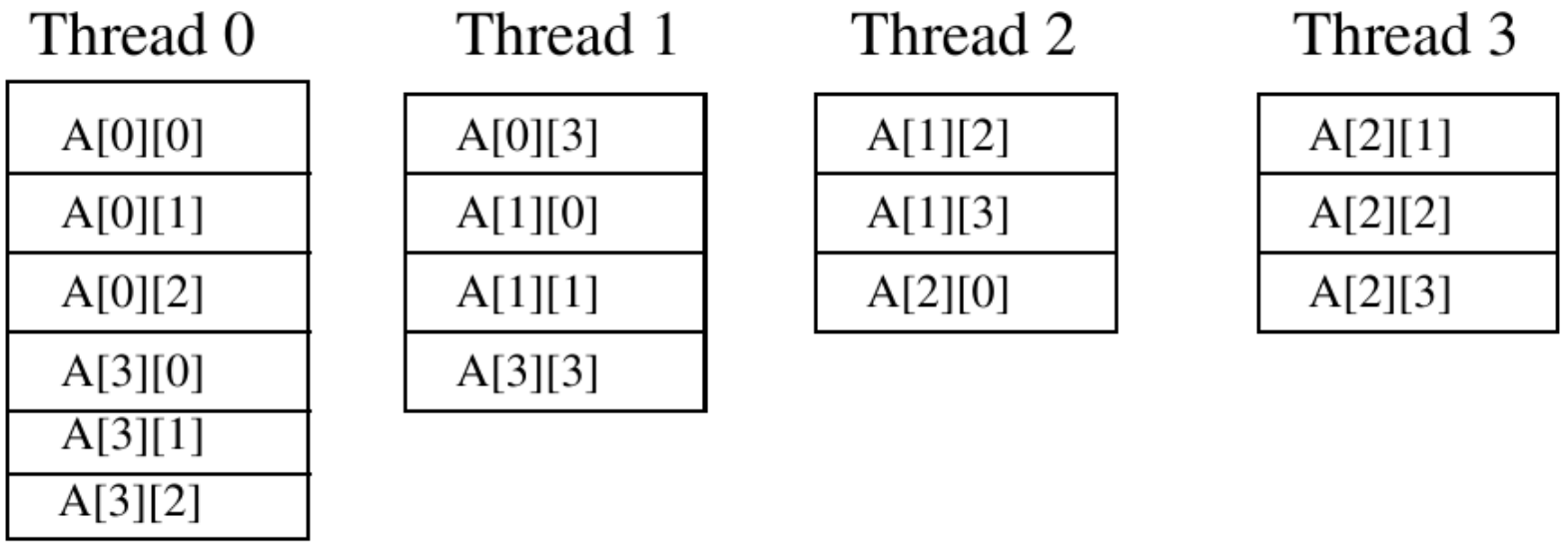
```
shared int A[4][THREADS];
```



UPC

- Poza domyślnym rozkładem tablic (po jednym elemencie) można wprowadzić rozkład blokami

```
shared [3] int A[4][THREADS];
```

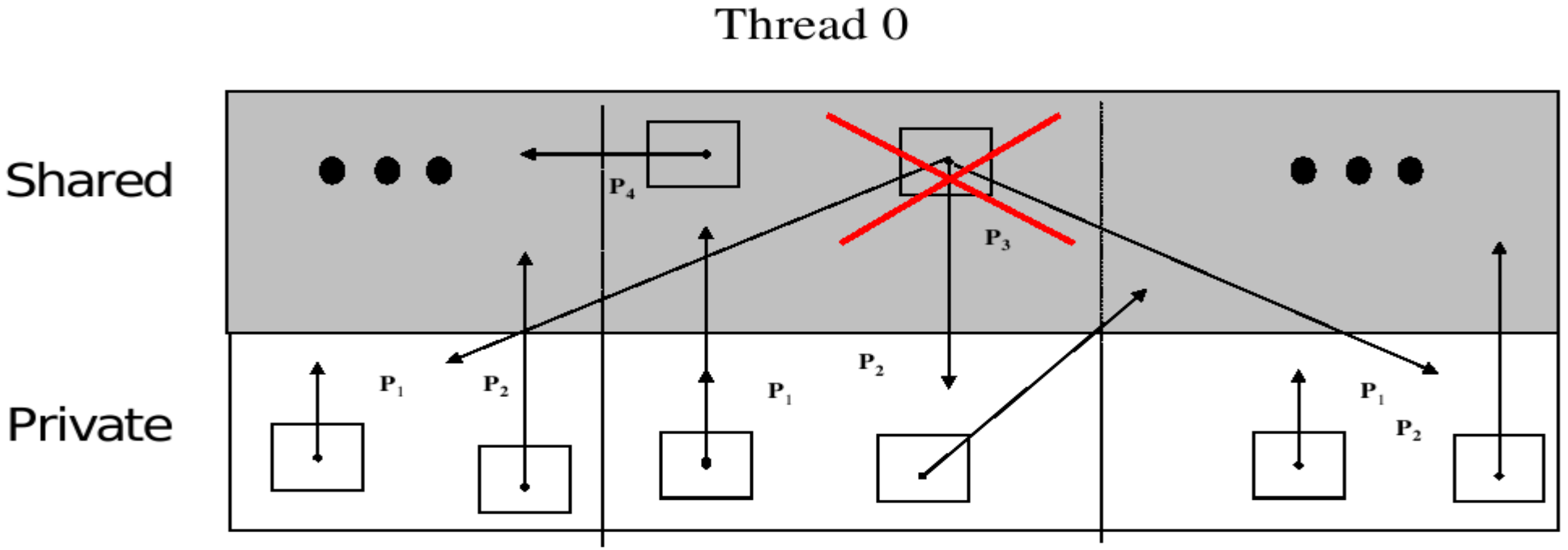


UPC

- Standardowe procedury języka umożliwiają kopiowanie danych z obszaru wspólnego do obszarów prywatnych wątków
- Istnieją procedury umożliwiające dynamiczna alokację tablic w obszarze pamięci wspólnej
- Wskaźniki do zmiennych mogą być:
  - prywatne – pokazują na elementy prywatne lub elementy z obszaru wspólnego o powinowactwie z wątkiem posiadającym wskaźnik
  - typu *pointer-to-shared* – wskazują na elementy w pamięci wspólnej

UPC

- int *p1; /* private pointer pointing locally */
- shared int *p2; /* private pointer pointing into the shared space */
- int *shared p3; /* shared pointer pointing locally */
- shared int *shared p4; /* shared pointer pointing into the shared space */



UPC

upc_forall(init; test; loop; affinity)

- Równoległa pętla for
  - *affinity* - oznacza powinowactwo dla każdej z iteracji:
    - jeśli jest liczbą związaną z indeksem pętli, iteracja jest przydzielana wątkowi: *affinity*@THREADS
    - jeśli odnosi się do egzemplarza danych (ewentualnie także zależnego od indeksu pętli), iteracja jest przydzielana wątkowi posiadającemu powinowactwo do tego egzemplarza

UPC

- Dwa przykłady skutkujące tym samym podziałem iteracji pomiędzy wątki:

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

```
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];
```

UPC

- Dwa przykłady skutkujące tym samym podziałem iteracji pomiędzy wątki:

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

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

UPC

- Przykład – mnożenie macierz-wektor w UPC:

```
#include <upc_relaxed.h>

shared [THREADS] int a[THREADS][THREADS];
shared int b[THREADS], c[THREADS];

void main (void) {
      int i, j;
      upc_forall( i = 0 ; i < THREADS ; i++; i)
{
            c[i] = 0;
            for ( j= 0 ; j< THREADS ; j++)
                  c[i] += a[i][j]*b[j];
      }
}
```

Alternatywne modele programowania

- OpenHMPP, OpenACC – dyrektywy dla programowania heterogenicznego (alternatywa dla OpenCL i CUDA)
- Chapel, Fortress, Co-Array Fortran, X10 – alternatywy PGAS dla UPC
- Ada, C++AMP – obiektowe języki i biblioteki równoległe
- Cilk – rozszerzenie wielowątkowe C/C++
- Erlang, Concurrent Haskell – i inne języki funkcjonalne
- SALSA – język oprogramowania zorientowanego na aktorów
- Linda – model programowania z pamięcią wspólną – przestrzenią „krotek”, *tuplespace*
- Sisal – model przepływu danych, *dataflow* (szereg innych języków, środowisk programowania, np. w grafice)
- i wiele, wiele innych