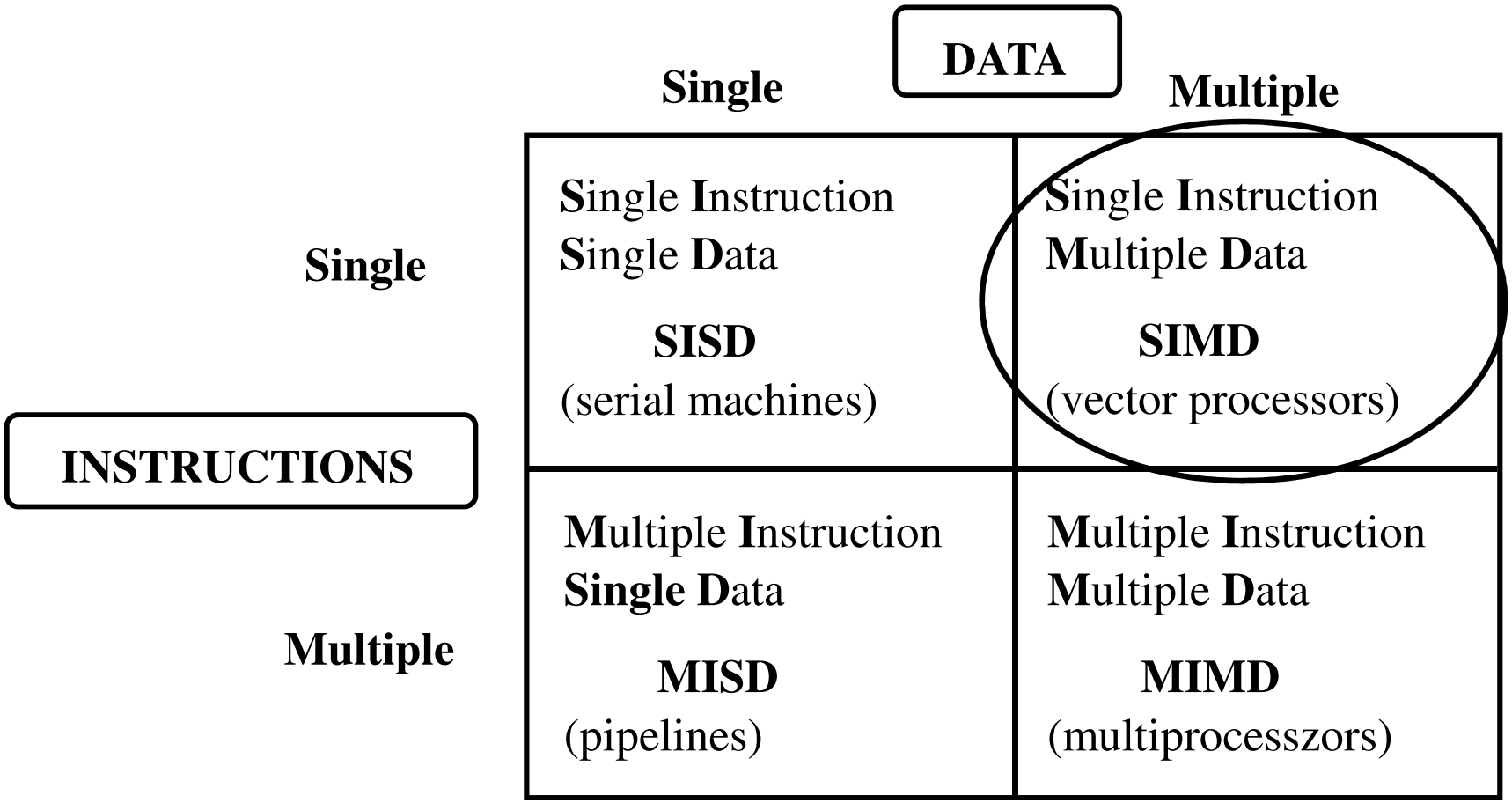
Párhuzamos és Grid rendszerek (10. ea)

egy vektorprocesszor

Szeberényi Imre
BME IIT

<szebi@iit.bme.hu>

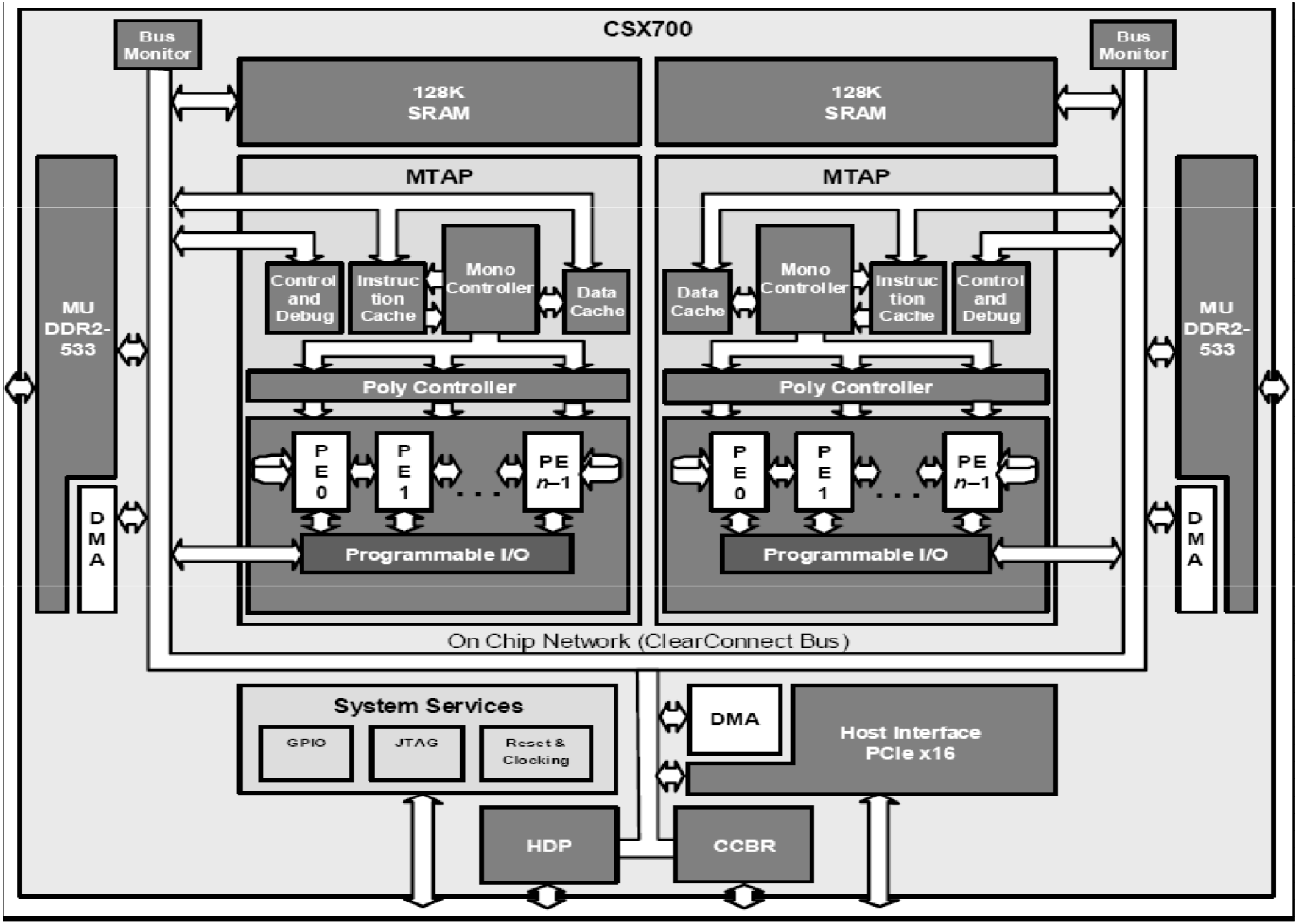
Flynn-féle architektúra modell



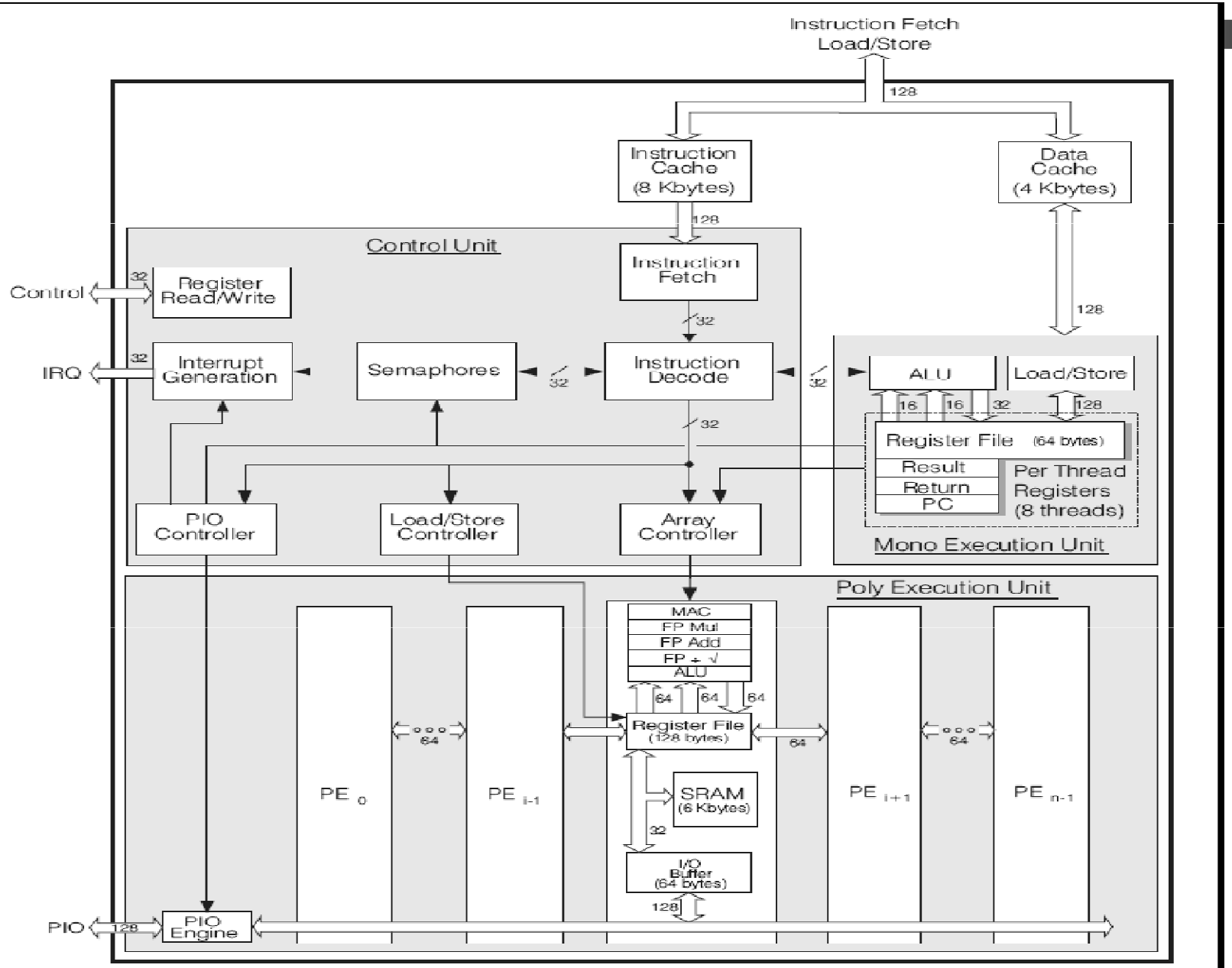
ClearSpeed gyorsító kártya

- 2006: Tokyo Institute of Technology's TSUBAME Supercomputer 47.38 TFLOPS
- ClearSpeed e710 kártya
  - Gyorsítókártya (PCIe x8)
  - 2 db 96 magos processzor
  - 2 x 8 GB DRR2, 2x128K SRAM
  - 96 Gflops
  - 24 W

Belső felépítés



Multi-threaded Array Processor

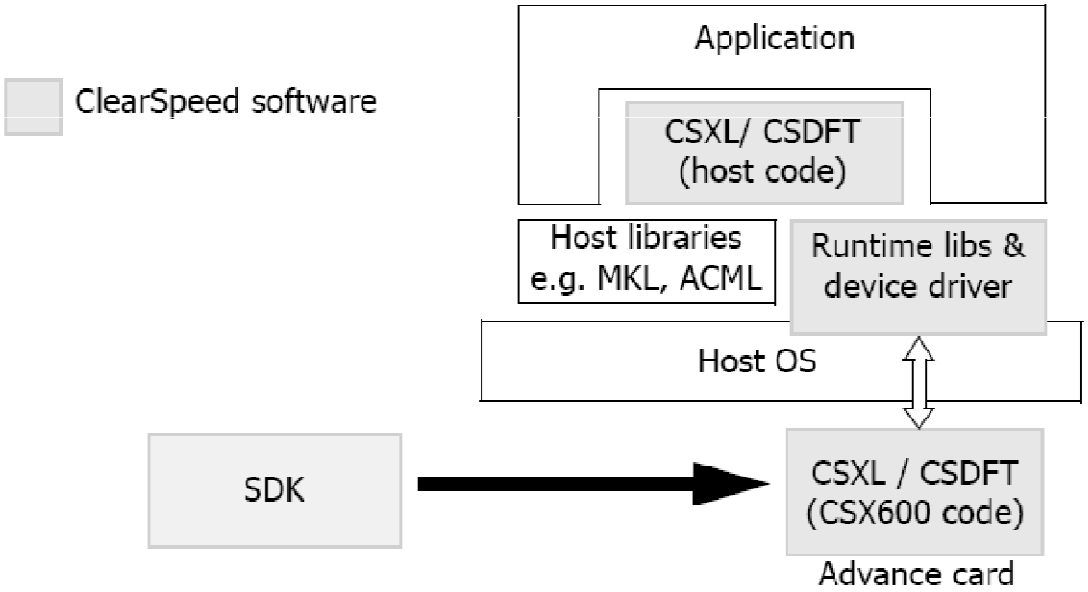


MTAP

- Control unit
  - fetch, decode, -> mono, v. poly
  - semaphore
- Poly controller
- Execution units
  - 1 db mono
  - 96 db poly
- Cache
- I/O

Szoftver komponensek, eszközök

- CSXL
  - BLAS, LAPACK
- CSAPI
- SDK
  - C^n nyelv
  - Standard lib
  - Compiler (gcc)
  - Debugger (gdb)
  - Szimulator



C^n nyelv

- Standard ANSI C +
  - 2 db új kulcsszó:
    - poly
    - mono
  - Szabályok a két új tárolási osztály elérésére, automatikus konverziójára.
    - pointerek kezelési szabályai
    - tömb, struct és union kezelési szabályai
    - vezérlési szerkezetek speciális értelmezése
  - Számos fv. a hw. kezelésére

Egyszerű példa

```
#include <stdio.h>
int main() {
  printf("Hello world\n");
  return 0;
}
```

Fordítás és futtatás:

cscn hello.cn –o hello.csx

csrun –r hello.csx

Kis módosítás

```
#include <stdiop.h>
int main() {
   printfp("Hello world\n");
   return 0;
}
```

mono és poly

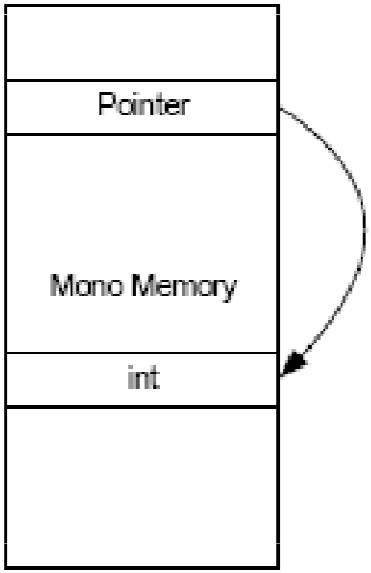
```
poly int counter;        // minden PE-ben
mono double data;        // csak a mono unit-ban
float fix;               // ua. mint mono
// poly visszatérési értékű függvény:
poly int fx(poly int, poly float);
// mono visszatérési értékű függvény
mono int fx(poly int, poly int);
```

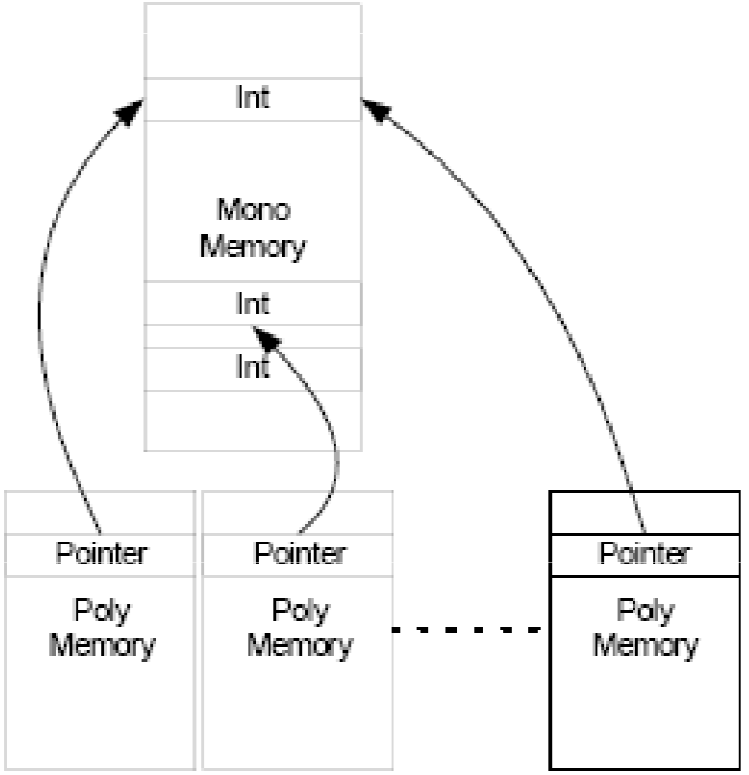
Újabb módosítás

```
#include <stdiop.h>
#include <lib_ext.h>
int main() {
  poly int penum = get_penum();
  printfp("Hello world %d\n",penum);
  return 0;
}
```

pointerek

mono int * mono p;

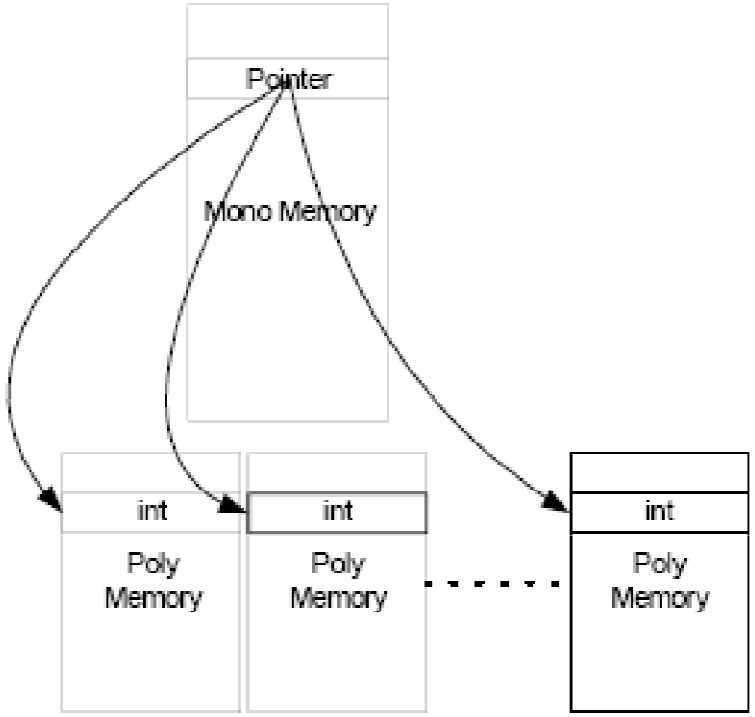


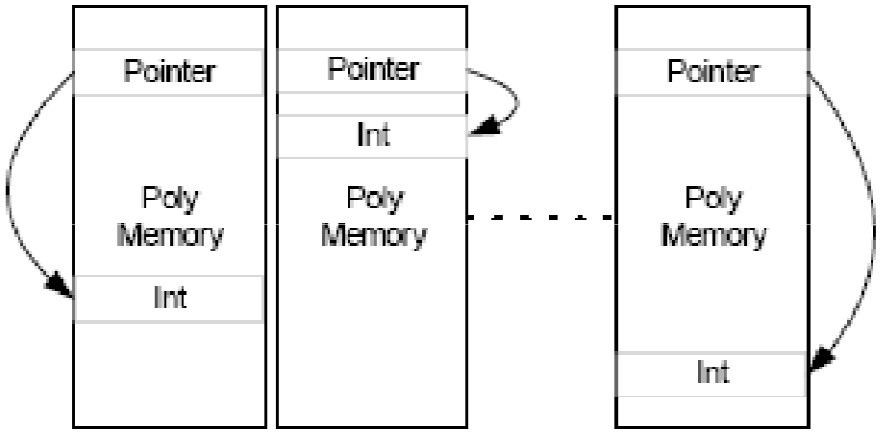


mono int * poly p;

pointerek /2

poly int* mono p;





poly int* poly p;

struct és union

- A multiplicitást csak a típuson keresztül lehet megadni.

```
struct str_t { int a; double d; };
mono struct str_t str1;
poly struct str_t str1;
```

- A typedef-fel is ugyanez a helyzet.

pointer cast és tömb

- nem lehet pointer multiplicitását megváltoztatni cast-tal! Másik memória terület!
- poly int tomb[100]
  - poly int* mono tomb;

vegyes multiplicitás

mono a; poly b;

- mono változó mindig konvertálható poly-vá. A konverzió automatikus.

  b = a;

- Kifejezésben is lehet vegyesen, a kifejezés értéke poly típusú lesz.

  a + b

- Értelmetlen, és nem is lehet mono változóba poly-t tenni.

vezérlési szerkezetek (if)

mono a; poly b;

if (a < 10) utas1 else utas2 // normál

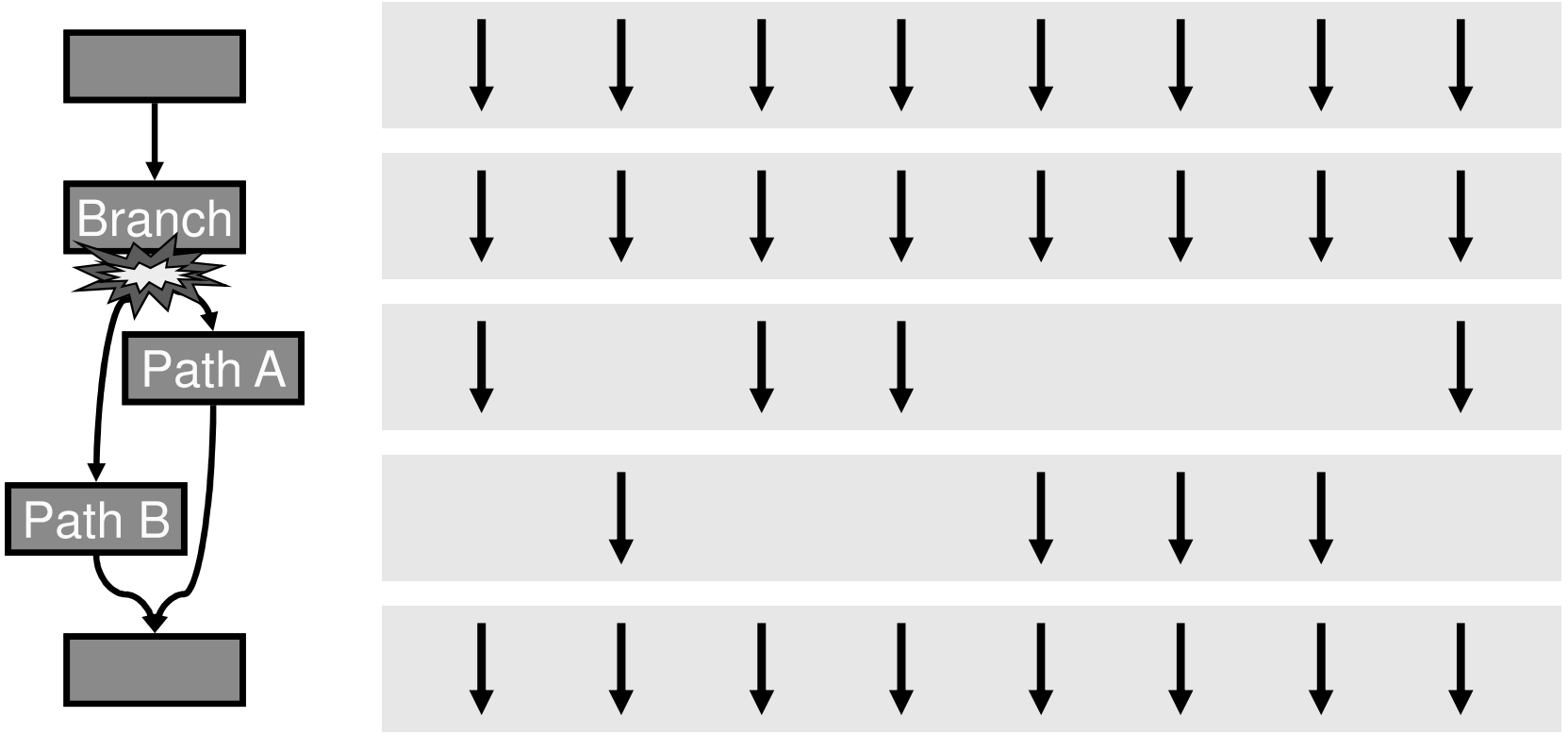
if (b < 10) utas1 else utas2 // minden PE-ben

- Azok a PE-k melyekben a kif. nem teljesül letiltódnak, majd az else végrehatásakor azok tiltódnak, amelyekben teljesült.
- Végül mindkét ág végrehajtódik.

```
if (b < 1) a = 5; else a = 8; // mennyi a ??
```

Vezérlés divergencia

Az gyakori elágazás rontja a párhuzamosítás hatékonyságát.



for, while, do

```
mono a;
for (a = 0; a < 10; a++) utas // normál
poly b, i; // tfh., hogy i minden PE-ben más
for (b = 0; b < i; b++) utas // minden PE-ben
```

- Azok a PE-k melyekben a kif. nem teljesül letiltódnak, a többiben fut a ciklus.

egyéb vezérlési szerkezet

- goto – nem léphet át poly kifejezésre alapozott feltételt!
- helyette címkézett break és continue használható:

```
for_i:
  for (i=...)  {
       if (...)
            break for_i;
  }
```

- switch csak a mono vezérlőben futhat!

Adatmozgatás

- memcpym2p
- memcpyp2m
- memcpym2p_strided
- memcpyp2m_strided

Van aszinkron változatuk is.

Használatuk figyelmet igényel.

szemaforok

- értéküк: nem negatív egész
- azonosítójuk: 0-92 (93-127: foglalt)
- műveletek:
  - sem_wait
  - sem_sig
  - sem_put
  - sem_get
  - sem_sync

1. példa: integrálás

```
double integ(double low, double high,
                                  long n) {
  double w, pew, sum; // mono-k
  poly int penum;     // PE sorszám
  poly double sump;   // PE részösszege
  poly double l, h;   // határok

  // PE sorszámának lekérdezése (0..95)
  penum = get_penum();

  w = (high-low)/n;   // lépésköz
  pew = (high-low)/96;// PE-nkénti lépés
```

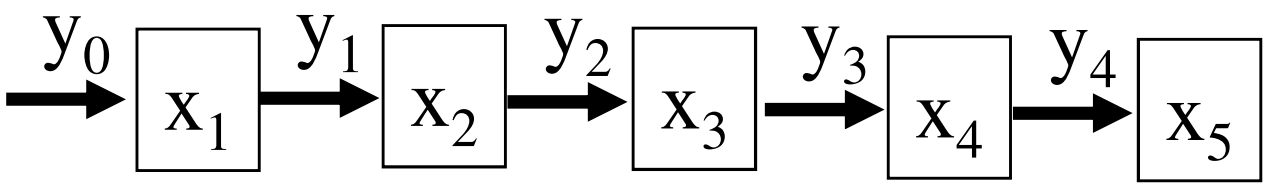
1. példa: integrálás / 2

```
  l = pew*penum+low; // alsó határ
  h = l + pew;       // felső határ
  sump = 0.0;        // téglányösszeg
  l += w/2.0;        // fél lépéssel
  while (l < h) {
    sump += fx(l);   // összegzés
    l += w;          // egész lépés
  }

  // minden PE sump-jét összeadja
  sum = cs_reduce_sum(sump);
  return sum * w;    // szorzás
}
```

1. példa: integrálás (PI)

```
#include <lib_ext.h>
#include <stdiop.h>
#include <stdlibp.h>
#include <reduction.h>

#define fx(x)    (4.0/(1.0+x*x))
#define N 1000000L
....
....
int main() {
  printf("PI(%10ld): %15.13lf\n", N,
  integ(0, 1, N));
  return 0;
}
```

2. példa: rendezés csővel



Minden feldolgozóra, minden lépésben:

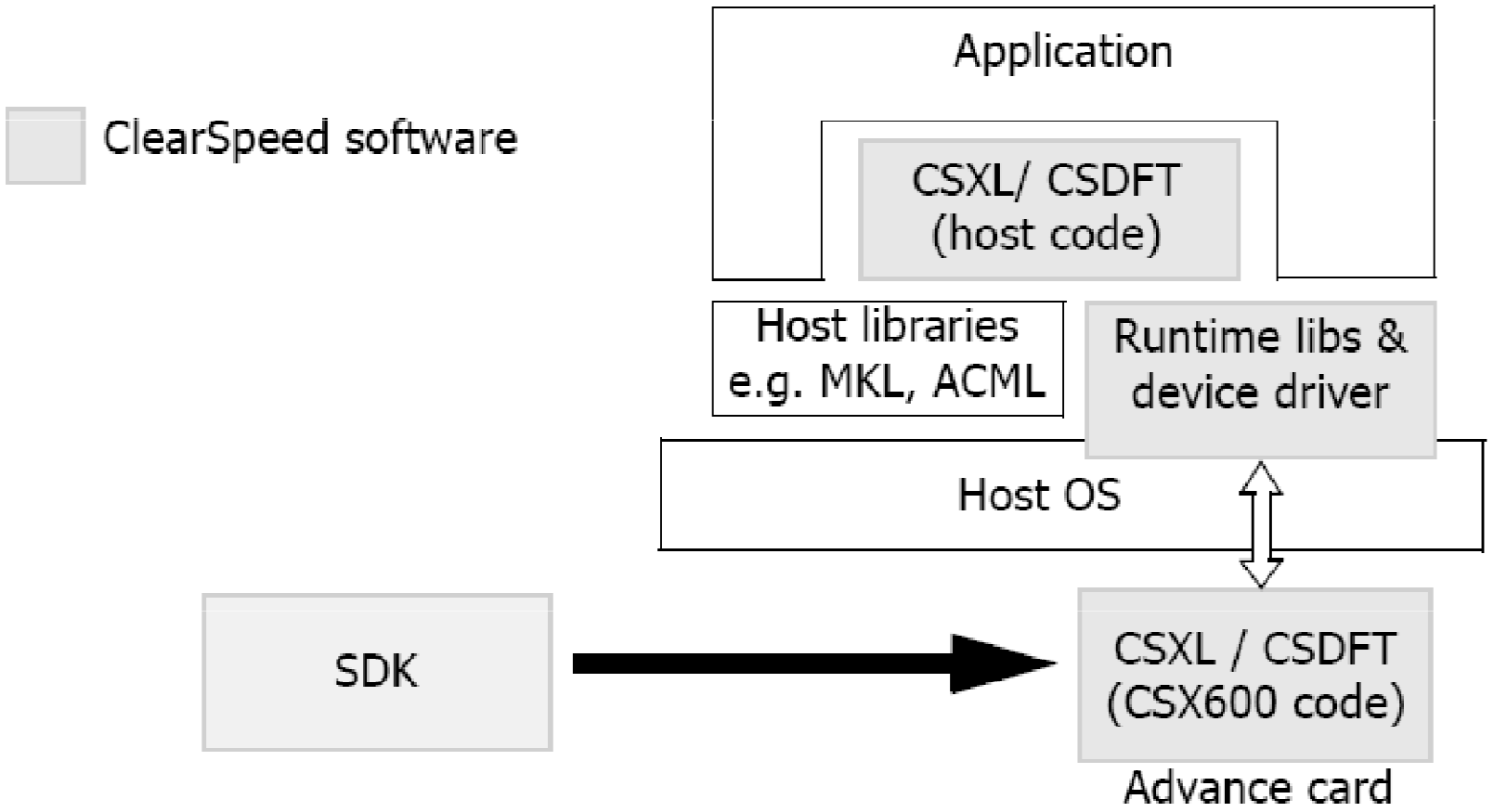
$x_i = \max(x_i, y_{i-1}),\ i = 1,2,3\ldots n$

$y_i = \min(x_i, y_{i-1}),\ i = 1,2,3\ldots n$

2. példa: rendezés csővel /2

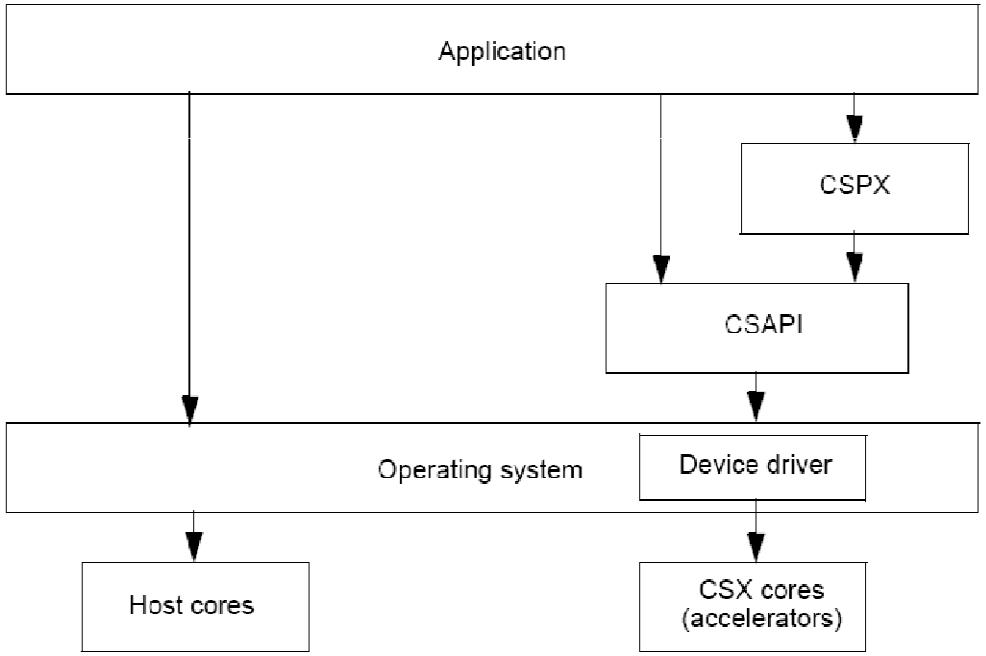
```
unsigned int vec[NUM_PES];
int i, j;
poly int x, y, tmp; // PE-kben

x = y = -1;              // speciális érték
for (i = 0; i < NUM_PES; i++) {
  set_swazzle_low(vec[i]); // új érték
  y = swazzle_up(y); // belép + előző y
  for (j = 0; j <= i; j++) { // i+1-szer
    if (y > x) {
      tmp = x; x = y; y = tmp;
    }
    y = circular_swazzle_up(y); // kisebb ->
  }
} // x-ek nagyság szerint csökkenően
```

Szoftver komponensek, eszközök

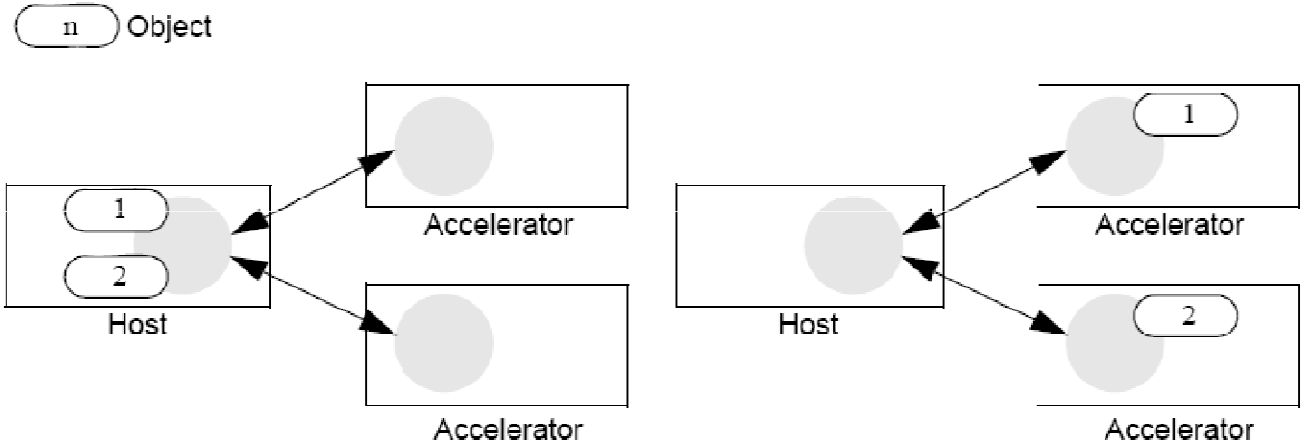


CSPX

- Device driver
- CSAPI
- CSPX
  - RPC
  - Processz + objektum modell



Processz és objektum



- CSPX::State<>
  - numProcesses, call, sync
- CSPX::Object<>
  - move
- CSPX::Parameter<>

Példa

- Vektor elemek négyzetgyöke
- Objektumok:
  - input, output vektor
- Processzek:
  - hoszt processz
  - gyorsító kártyák processzei

Példa (cpp #1)

```
int main(void) {
  static double inp[SIZE], res[SIZE];

  CSPX::State<> processes("sqrt.csx");

  int numProc = processes.numProcesses();// CPU-k

  CSPX::Object<double> inputObj(inp, SIZE,
                    numProc, CSPX_OBJECT_WRITE);
  CSPX::Object<double> resultObj(res, SIZE,
                    numProc, CSPX_OBJECT_READ);
  .........
}
```

Példa (cpp #2)

```
int main(void) {
....
  CSPX::Parameter pars(numProc);
  pars.push(inputObj);  // sorrend ua. mint a
  pars.push(resultObj); // cn prog struktúrában
  pars.push(SIZE/numProc);

  for (int i = 0; i < SIZE; i++)
     inp[i] = i*i;
  processes.move(inputObj);
  processes.move(resultObj);
....
}
```

Példa (cpp #3)

```
int main(void) {
   ....

   processes.call(pars, "square_root"); // RPC!

   processes.sync(resultObj);

   for (int i=0; i < SIZE; i++) {
         printf("%d: %.2f\n", i, res[i]);
   }
}
```

Példa (cn #1)

```
struct sqrt_args {          // ebben a sorrendben
    CSPXObject inputs;      // ment a pars.push
    CSPXObject results;
    int count;
};
#pragma cspx_export(square_root) // elérhető lesz
int square_root(CSPXProcess parent, void* pars) {
  struct s_args *arg = (struct s_args *) pars;
  double *inp, *res;
  int i; CSPXErrno e;
  CSPX_object_sync(&args->inputs); // átveszi
  CSPX_object_sync(&args->results);
....
}
```

Példa (cn #2)

```
int square_root(CSPXProcess parent, void* pars)
   ...
  inp = CSPX_object_get_pointer(&arg->inp, &e);
  res = CSPX_object_get_pointer(&arg->res, &e);
  for (i = 0; i < args->cnt; i += get_num_pes()){
    poly double d;
    poly int index;
    index = i + get_penum();
    memcpym2p(&d, &inp[index], sizeof(double));
    d = sqrtp(d);
    memcpyp2m(&res[index], &d, sizeof(double));
  }
  CSPX_object_move(parent, &args->results);
}
```