Memory Access Reconstruction Based on Memory Allocation Mechanism for Source-Level Simulation of Embedded Software

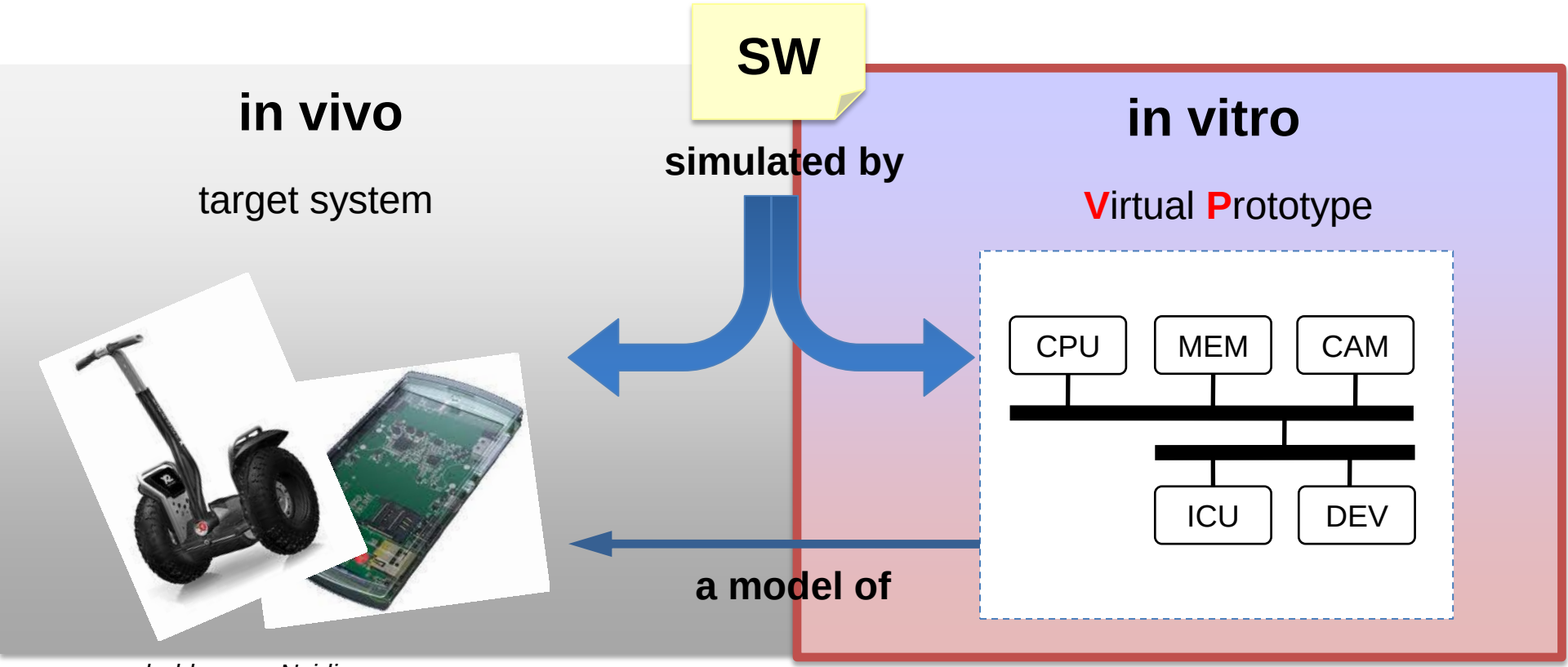
Kun Lu, Daniel Müller-Gritschneder, Ulf Schlichtmann
Lehrstuhl für Entwurfsautomatisierung
Technische Universität München

Gefördert durch BMBF Projekt SANITAS
(Förderkennzeichen. 01 M 3088)

Outline

- Background and motivation
- Related work
- Our apporach
  - **Based on memory allocation principles**
  - **Reconstruct memory accesses for cache simulation**
- Experimental results

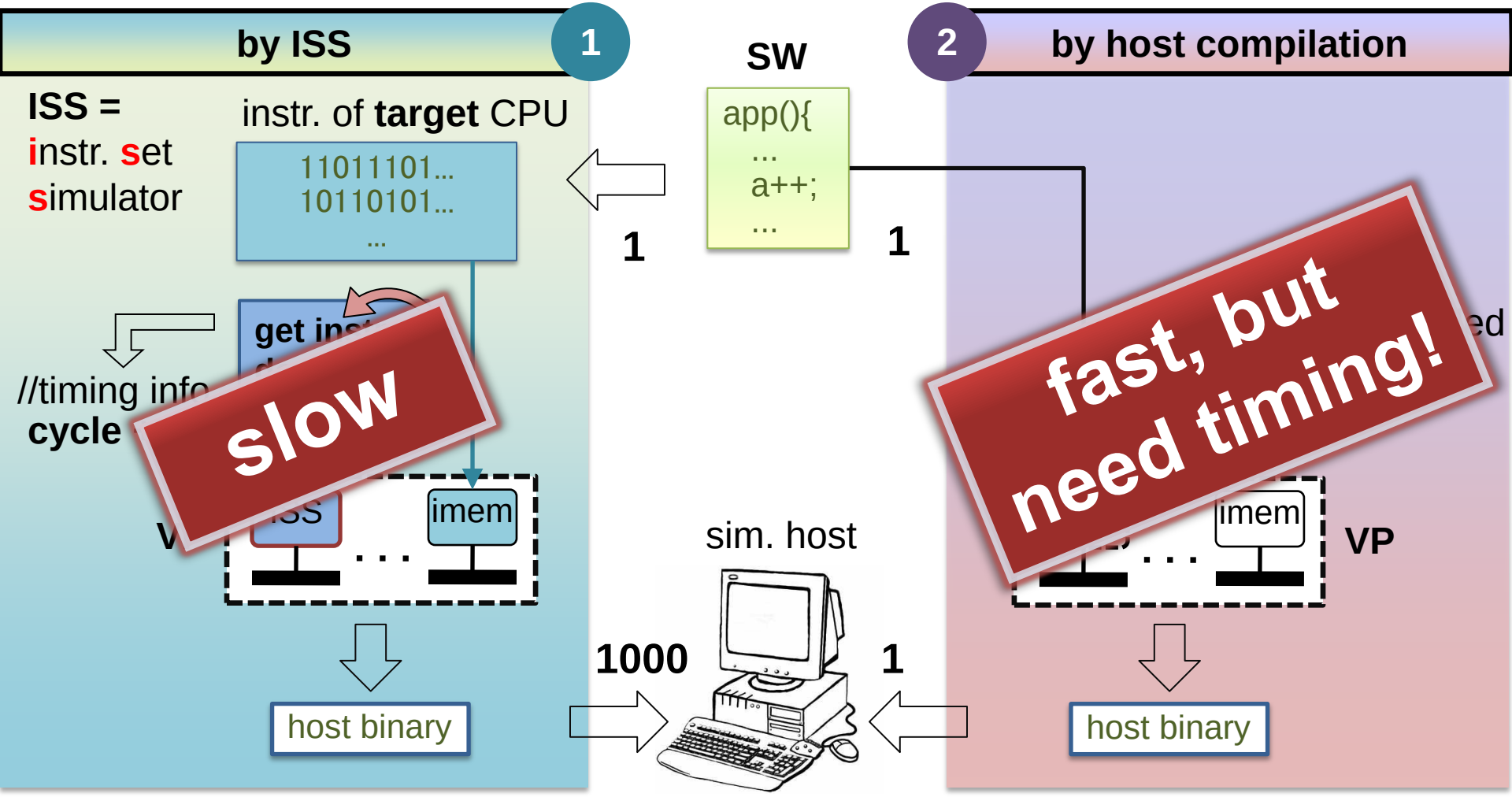
Background – Use VPs for SW development



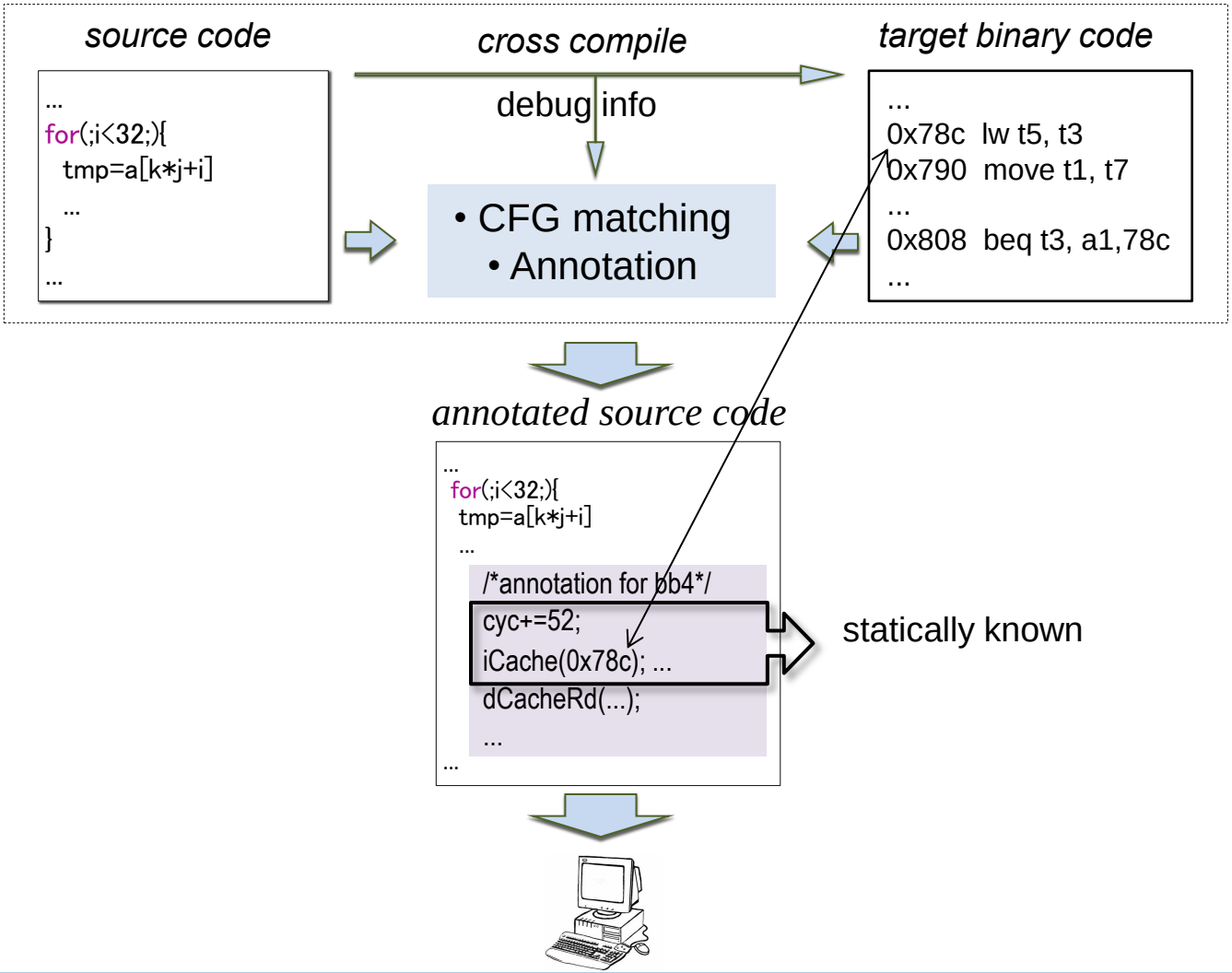
source: mashable.com, Nvidia

- **HDL, e.g., SystemC**
- ✓ **early** development, integration, verification

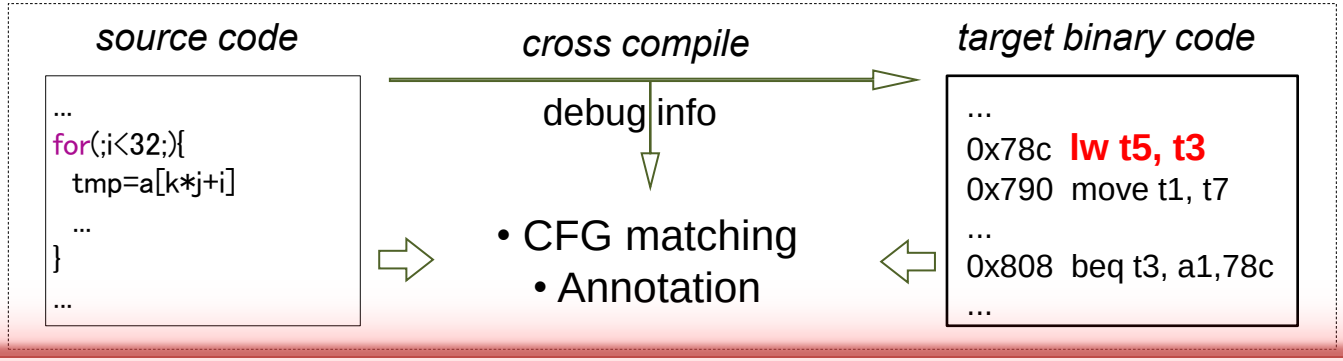
Background – SW simulation with VP



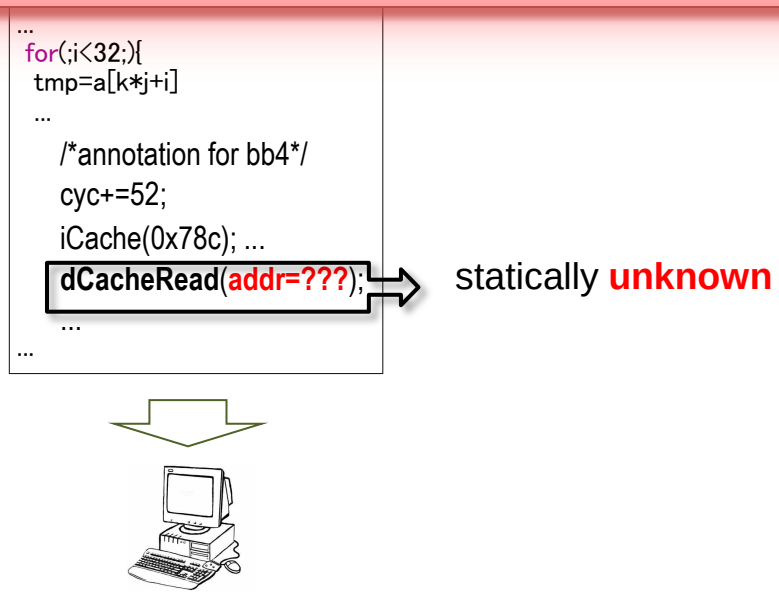
Background - Host-compiled SW simulation



Background - Host-compiled SW simulation



Problem: data memory Rd/Wr addresses unknown!



Related work

- Some get around the problem - no data cache simulation
- Random cache miss for each memory access [Hwang, DATE'08, Lin, ASP-DAC'10]
- Consider only the global/static data [Pedram, IESS'09]
- Use host-machine address emulation [Kempf, DATE'06, Posadas, ASP-DAC'10]
  - Not all memory accesses can be emulated (e.g. those related to register spilling)
  - different data locality in host-machine and the target machine => inaccurate data cache simulation
- Worst-case address range [Stattelmann, DATE'12]

Problem still UNSOLVED

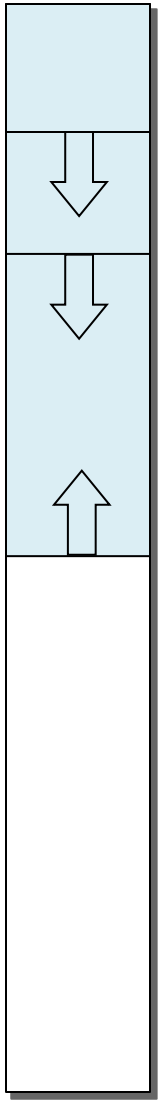
Our solution:

Exploit the Memory Allocation Mechanism

- Hit the nail on the head

Basic memory allocation principles

Stack



sp (stack pointer)

function call

sp (stack pointer)

function call

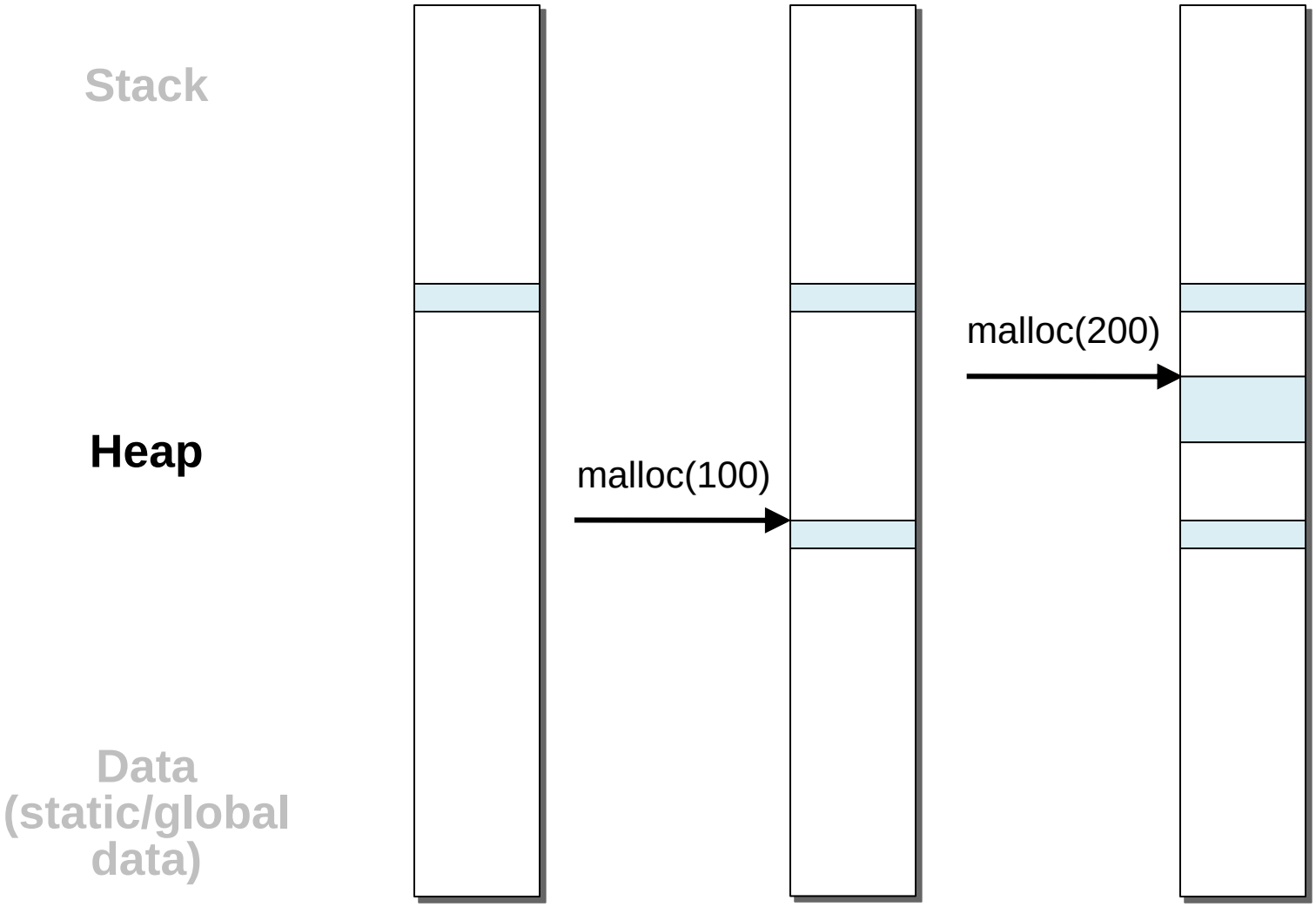
function return

sp (stack pointer)

Heap

Data (static/global data)

Basic memory allocation principles

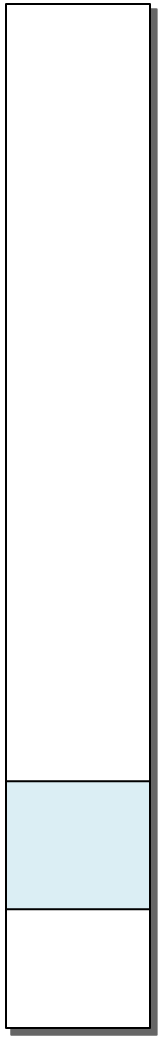


Basic memory allocation principles

Stack

Heap

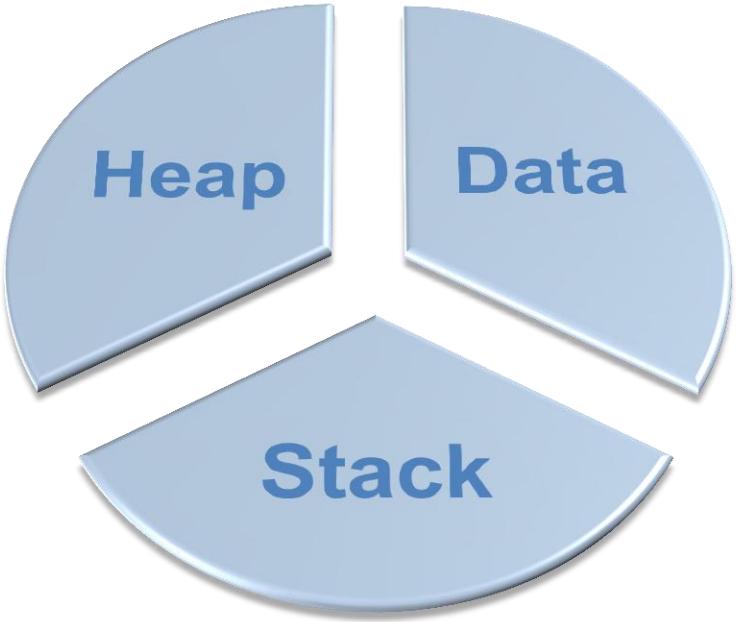
Data (static/global data)



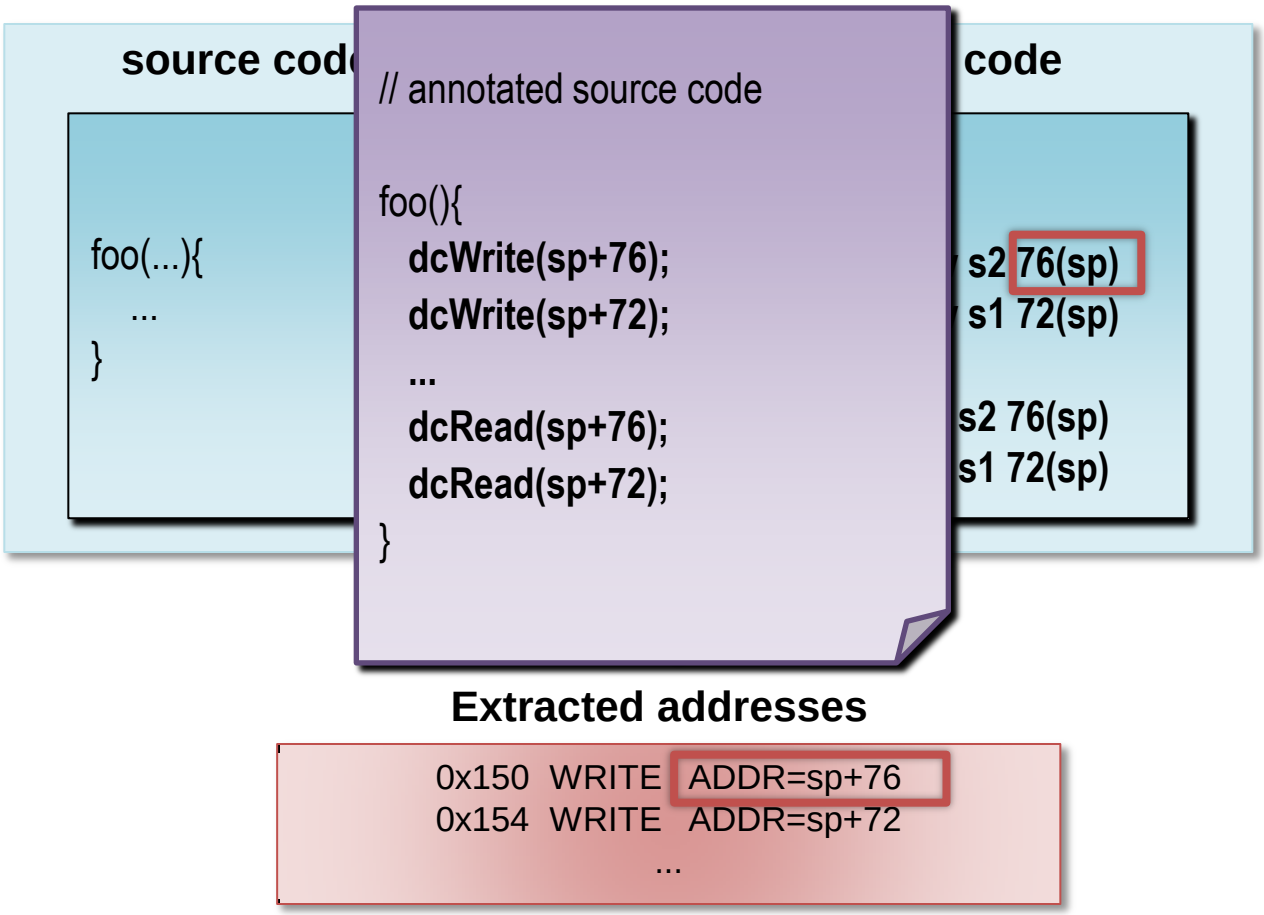
Allocated at compile time => fix address

Memory access addresses reconstruction

- handle each case



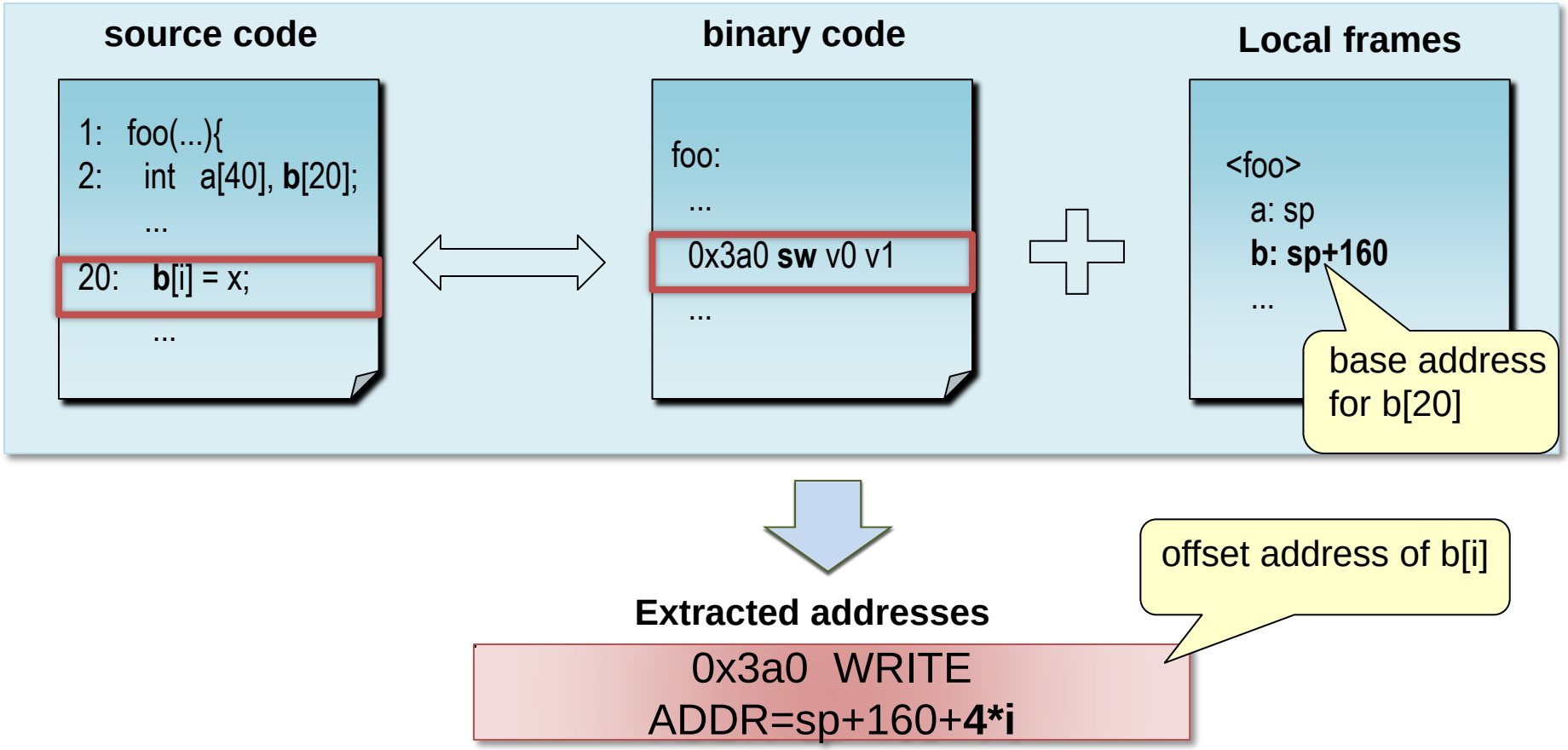
- When life is simple – the address is **sp-explicit**



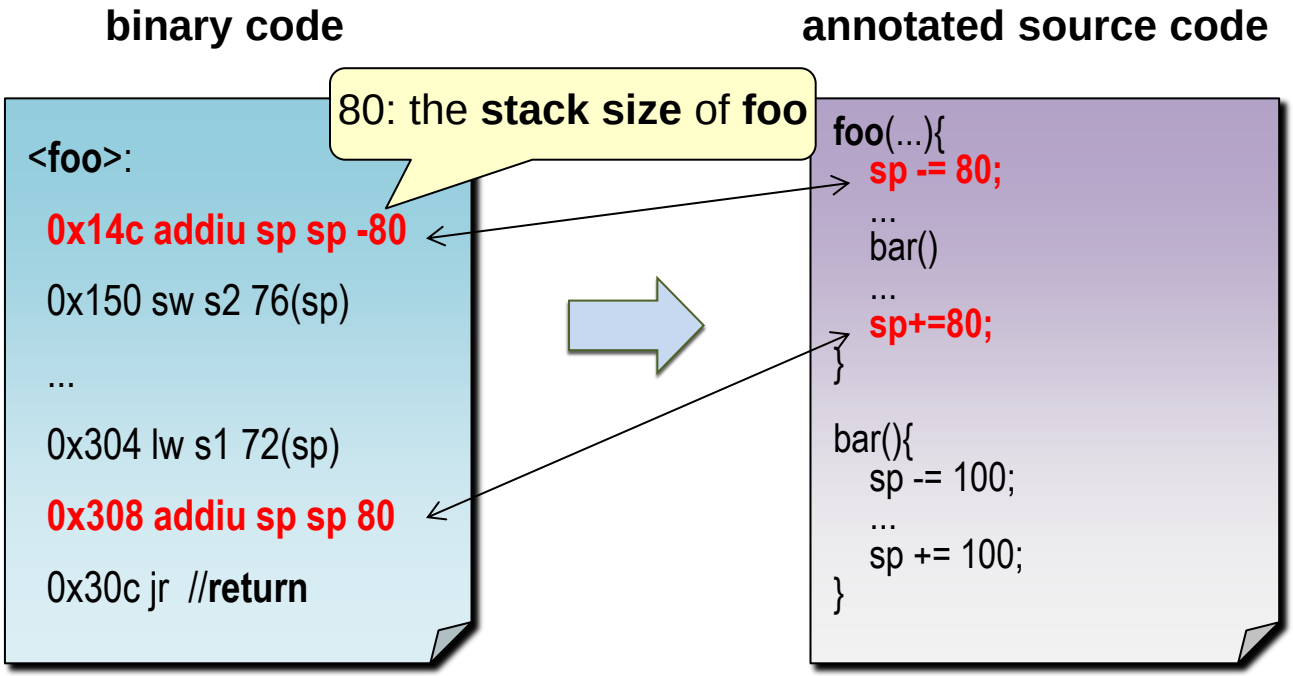
Stack

Heap

Data



- "**sp**" simulation



source code

```
A(){
   ...
   ptr =  malloc(100);
   for(){
      ptr[i] = ...;
   }
}
```

annotated source code

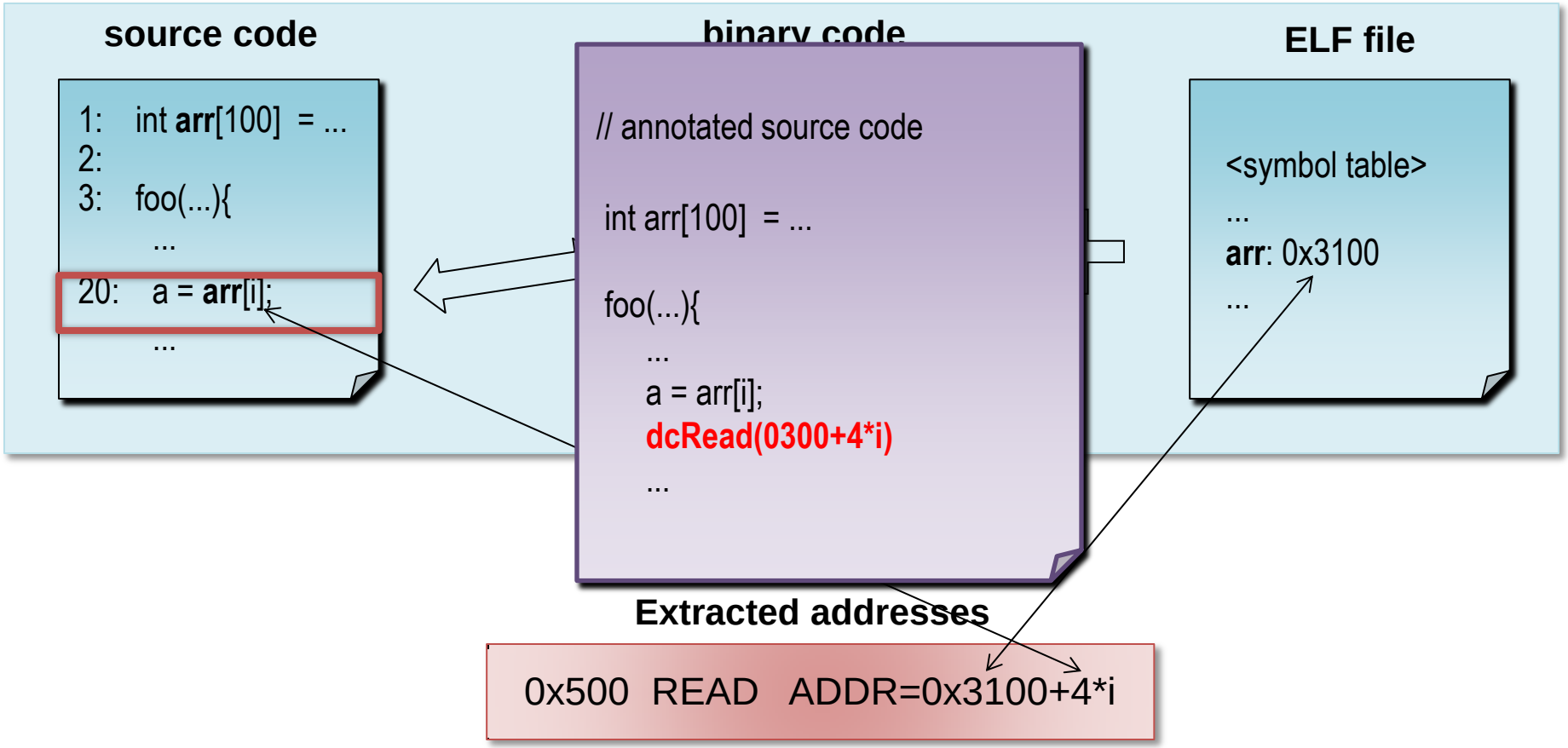
```
#define malloc OS_Malloc

A(){
   ...
   ptr =  malloc(100);
   for(){
      ptr[i] = ...;
      dcWrite(ptr);
      ...
   }
}
```

OSHeap.c

```
OS_Malloc(){
   ...
}

OS_Free(){}
```

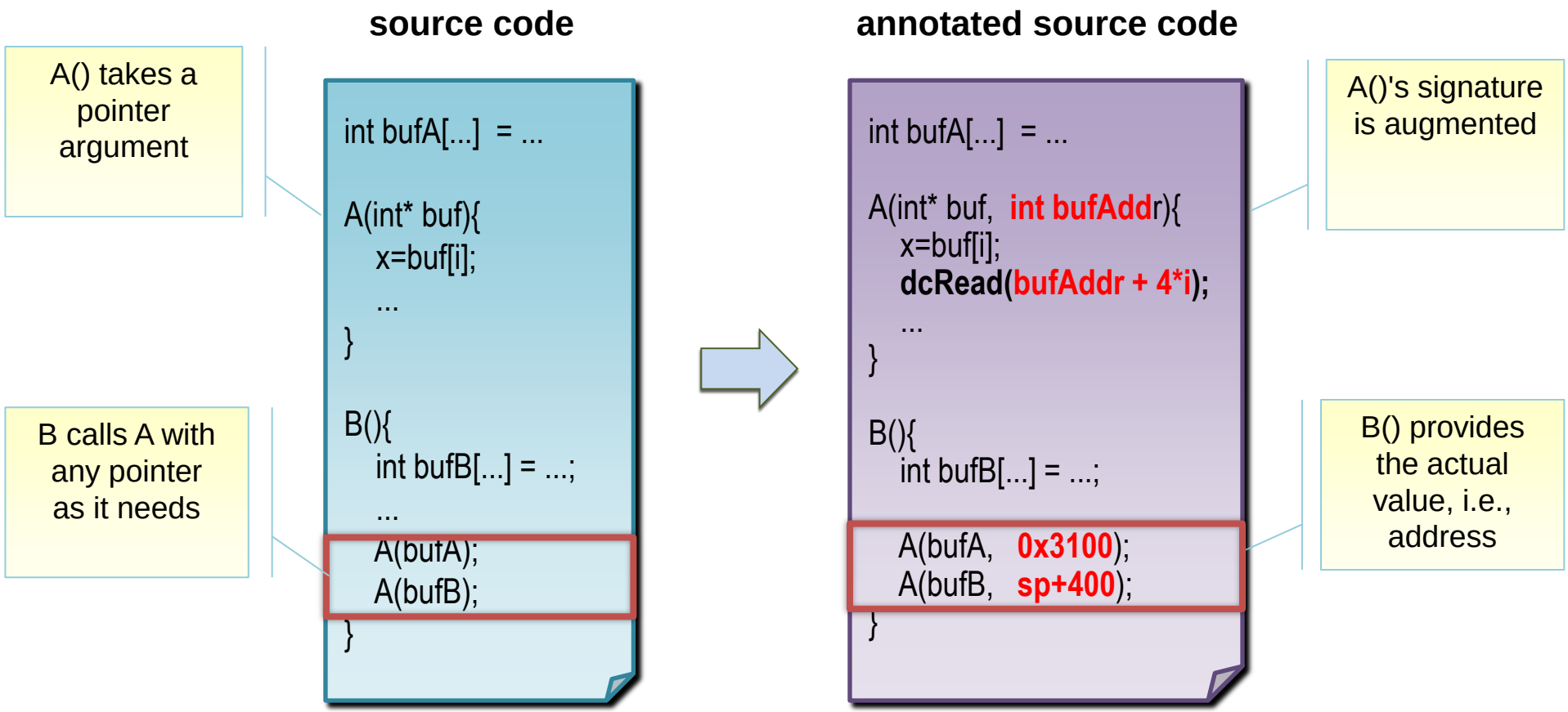
Directly used as the address, since heap allocation is simulated

Stack Heap **Data**



Handel pointers – 1/2

- Pointers used as function arguments

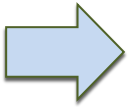


Handel pointers – 2/2

- Pointer arithmetic

source code

```
char A[100];
char B[100];

A(){
   if(...) {ptr=A;}
   else {ptr=B;}

   for(...){
      x=ptr[0];
      ptr++;
      ...
   }
}
```



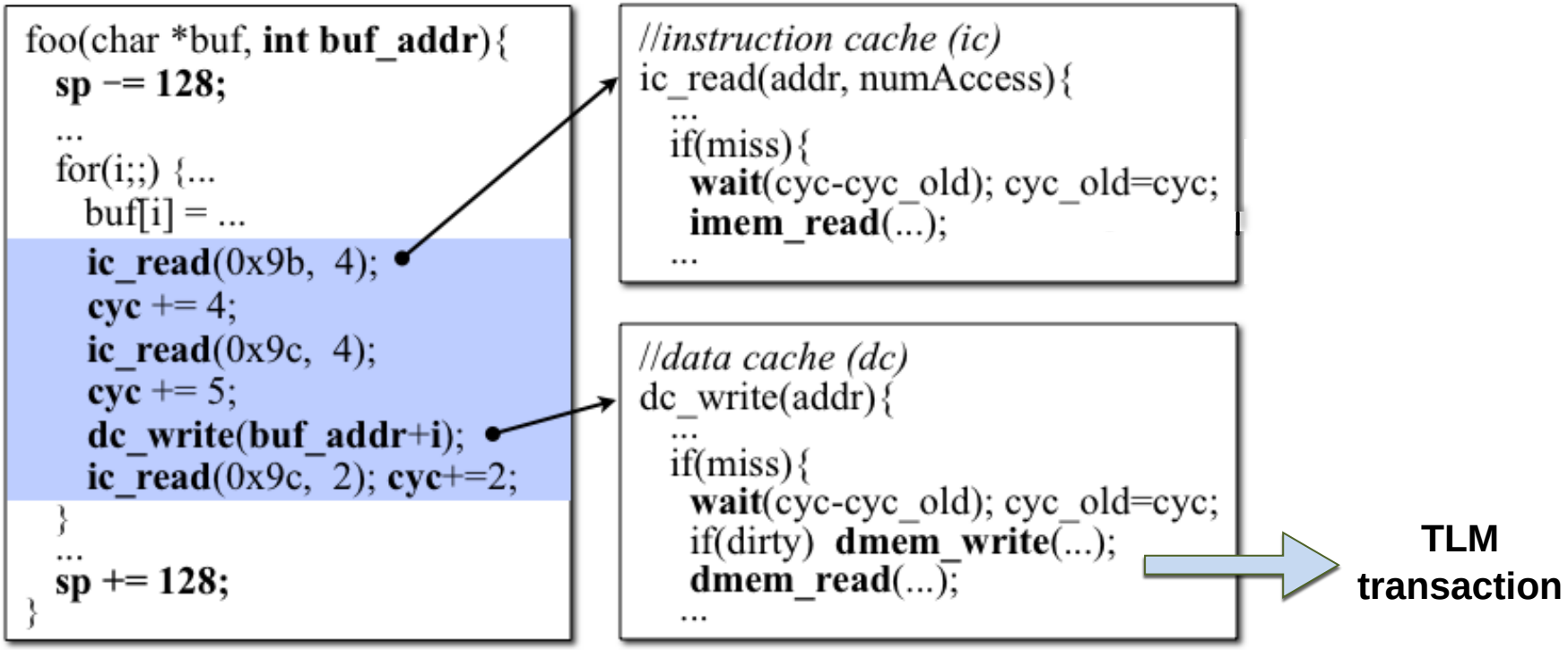
annotated source code

```
char A[100];
char B[100];

A(){
   if(...) {ptr=A; ptr_addr=A_addr; }
   else  {ptr=B;  ptr_addr=B_addr;  }

   for(...){
      x=ptr[0];
      dcRead(ptr_addr);
      ptr++; ptr_addr+=1;
      ...
   }
}
```

Use ptr_addr to trace the pointer address

Update the ptr_addr according to the pointer arithmetic

An example of the instrumented source code

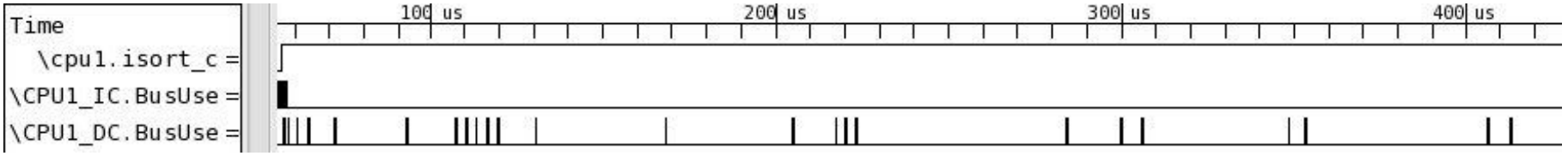


Experimental results – benchmark simulaiton

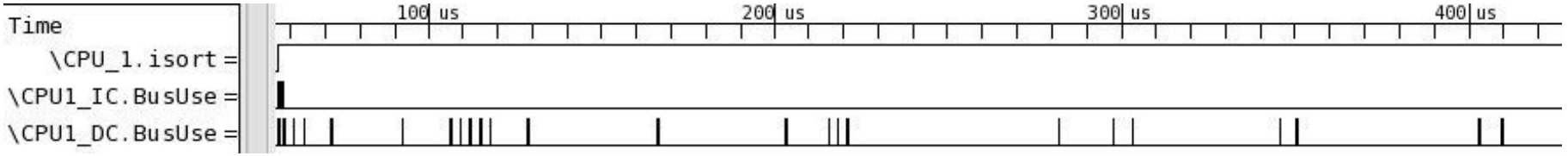
	N_{cycles}			i-cache: N_{access}/N_{miss}		d-cache: $N_{read}/N_{write}/N_{miss}$	
SW	ISS	SLS	error(%)	ISS	SLS	ISS	SLS
fir	233939	233448	-0.2	189980/13	189731/13	30254 / 1903 / **424**	30254 / 1902 / **427**
iir	98590	98481	-0.1	76229/13	76226/13	15257 / 5256 / **71**	15257 / 5256 / **69**
jpegdct	97418	97475	0.06	66877/45	66895/45	16261 / 11256 / **99**	16261 / 11203 / **100**
isort	89910	89996	0.1	73139/6	73177/6	8146 / 7953 / **26**	8152 / 7953 / **27**
r2y_malloc	134159	132919	-0.92	114772/19	114772/18	2057 / 2063 / **517**	2057 / 2061 / **518**
aes	12896	12867	0.22	7551/71	7564/70	1665 / 1160 / **49**	1676 / 1159 / **51**
edgeDetect	1050008	1050527	0.05	879263/16	880443/15	155019 / 8397 / **281**	155081 / 8396 / **279**

Experimental results – vcd traces of cache misses

isort – ISS simulation

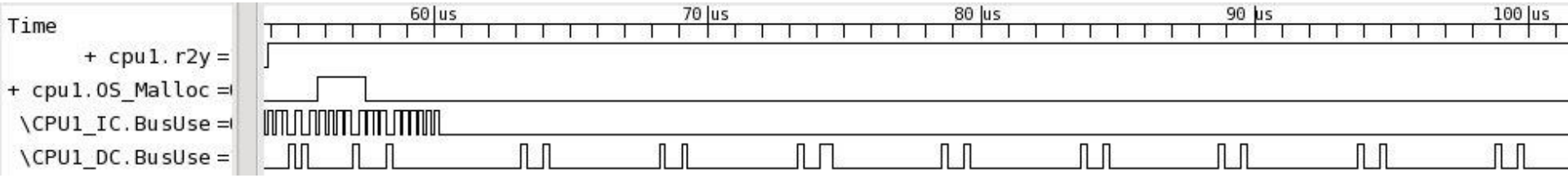


isort – Host-compiled simulation

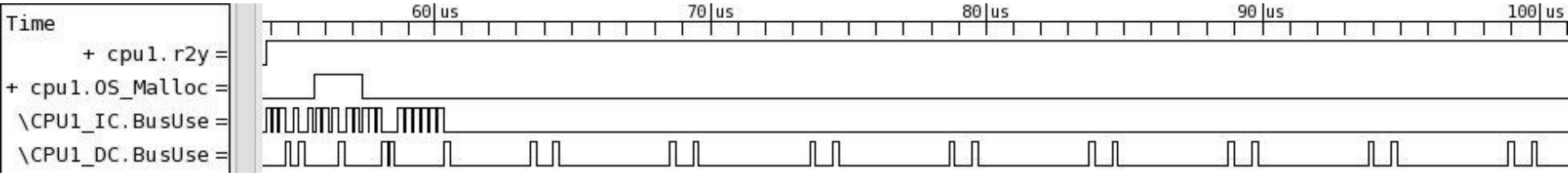


Experimental results – vcd traces of cache misses

Rgb2Yuv – ISS simulation

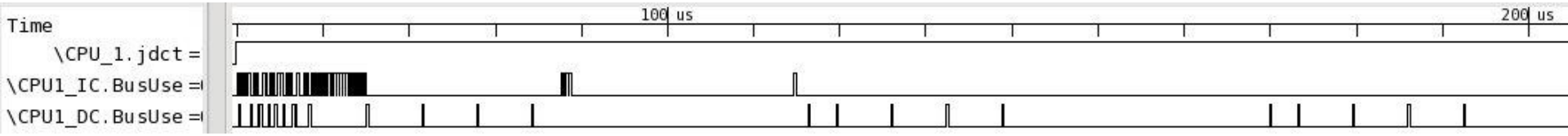


Rgb2Yuv – Host-compiled simulation

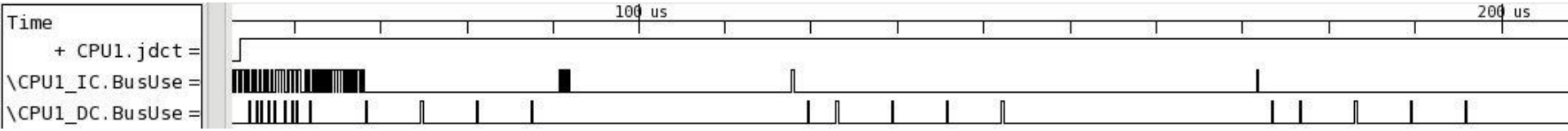


Experimental results – vcd traces of cache misses

JpegDCT – ISS simulation

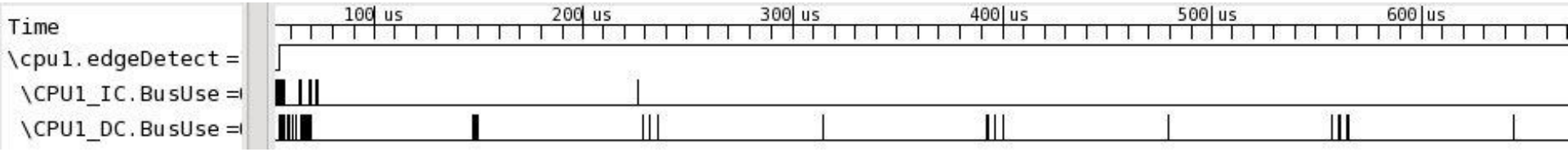


JpegDCT – Host-compiled simulation

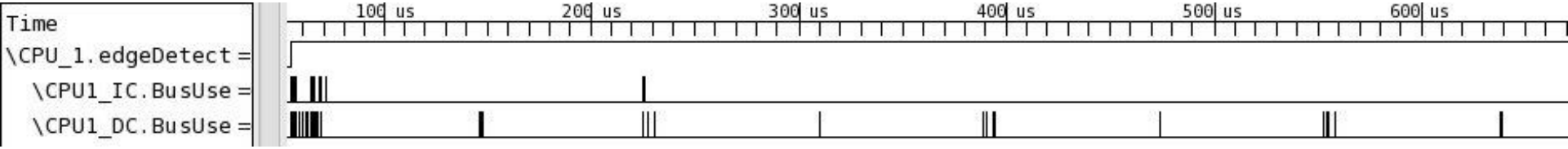


Experimental results – vcd traces of cache misses

EdgeDetect – ISS simulation



EdgeDetect – Host-compiled simulation



Conclusion

- For compiled SW simulation: memory addresses extracted by exploiting memory allocation mechanism
  - data cache simulation made possible
  - enables TLM simulation
  - ensure overall cycle accuracy

Thank you!