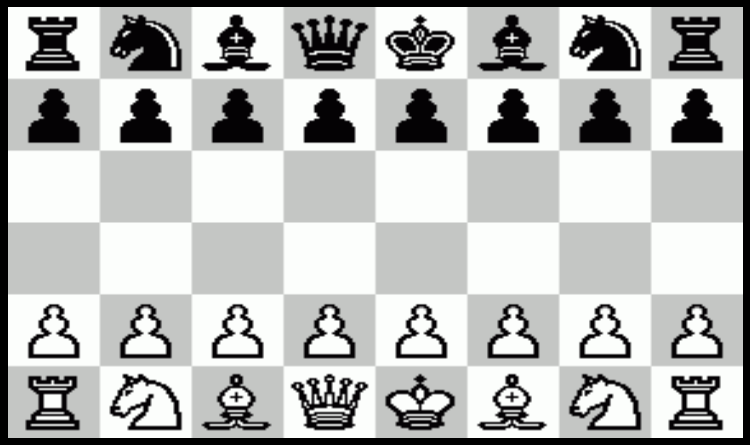
Architecture-aware Automatic Computation Offload for Native Applications

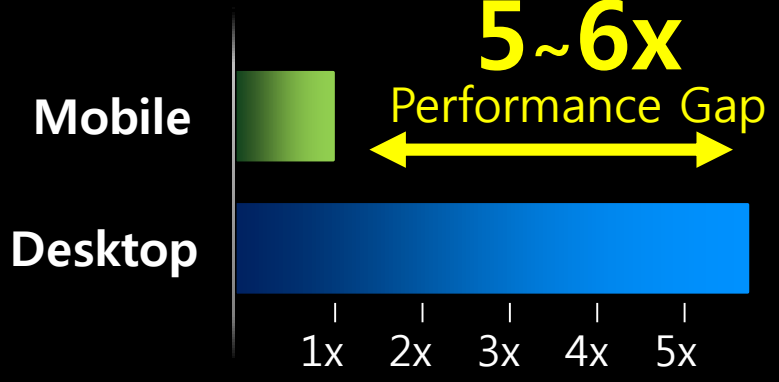
Gwangmu Lee, Hyunjoon Park, Seonyeong Heo,
Kyung-Ah Chang*, Hyogun Lee*, and Hanjun Kim.

POSTECH / Samsung Electronics*

Mobile devices are slow

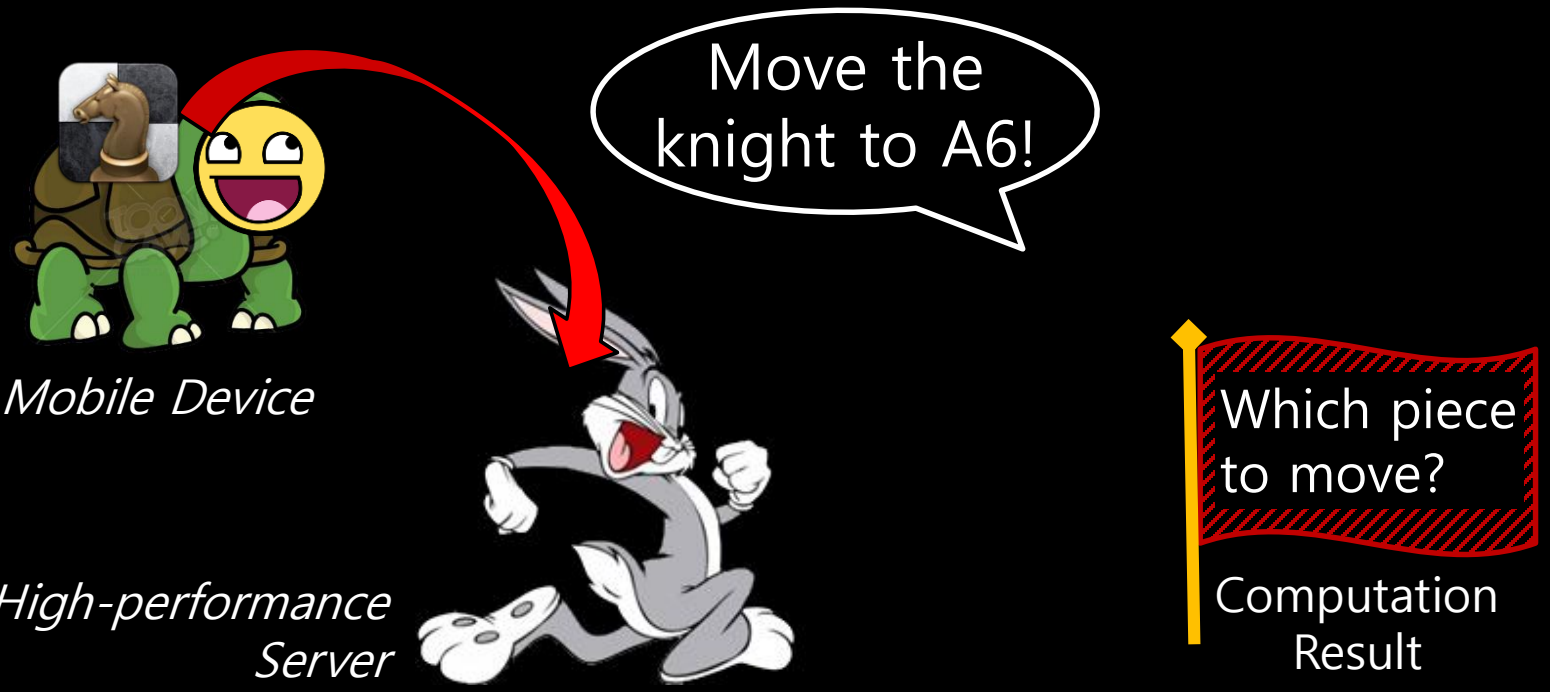
```
void runGame () {
  while (!gameover) {
    Move mv;

    /* User Inputs */
    mv = getPlayerTurn ();
    pieces[mv.tar] = mv.to;

    /* Heavy Computation */
    mv = getAITurn ();
    pieces[mv.tar] = mv.to;
  }
}
```



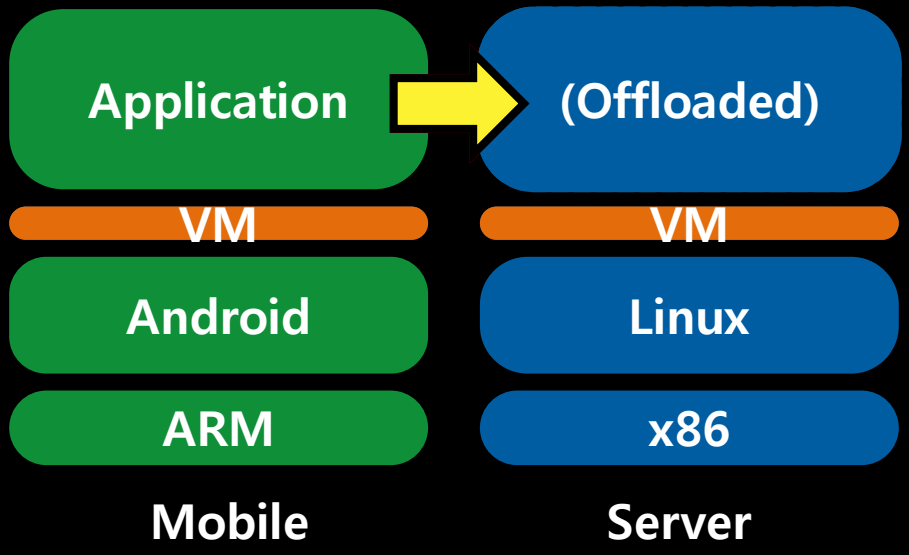


Chess Movement Computation

Offloading can boost your mobile device!

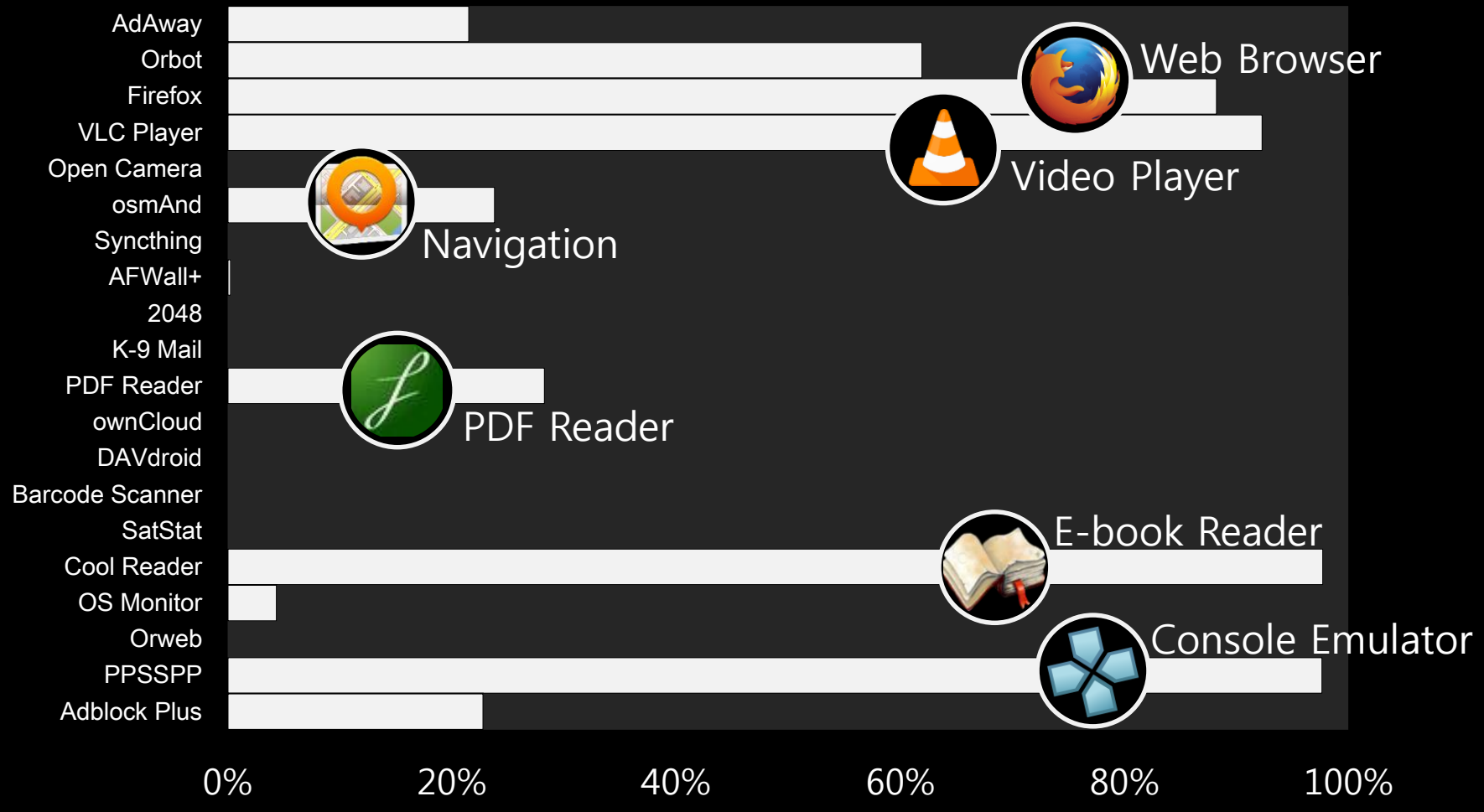


Most offloading systems are based on VMs.

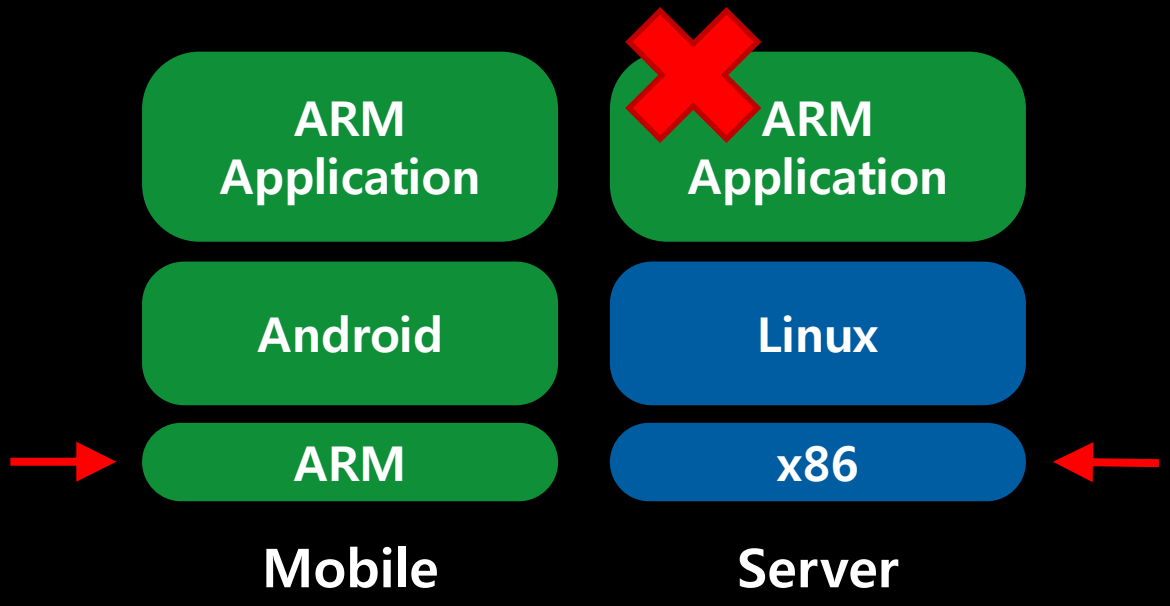


CloneCloud (EuroSys`11), **MAUI** (MobiSys`10), **CMCloud** (CCGrid`14)

Native Code Execution Time of Top 20 Android Applications



Challenges in offloading native workloads

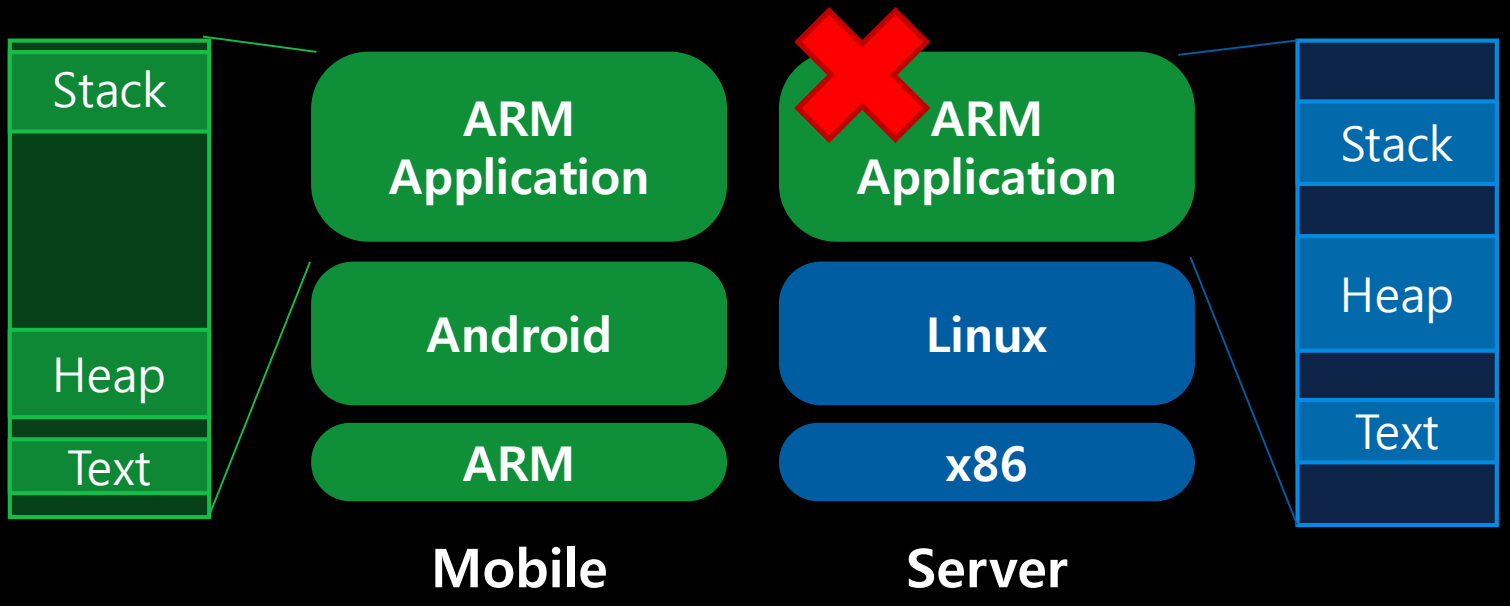


Different Architecture

Distinct Memory

Different Memory Layouts

Challenges in offloading native workloads

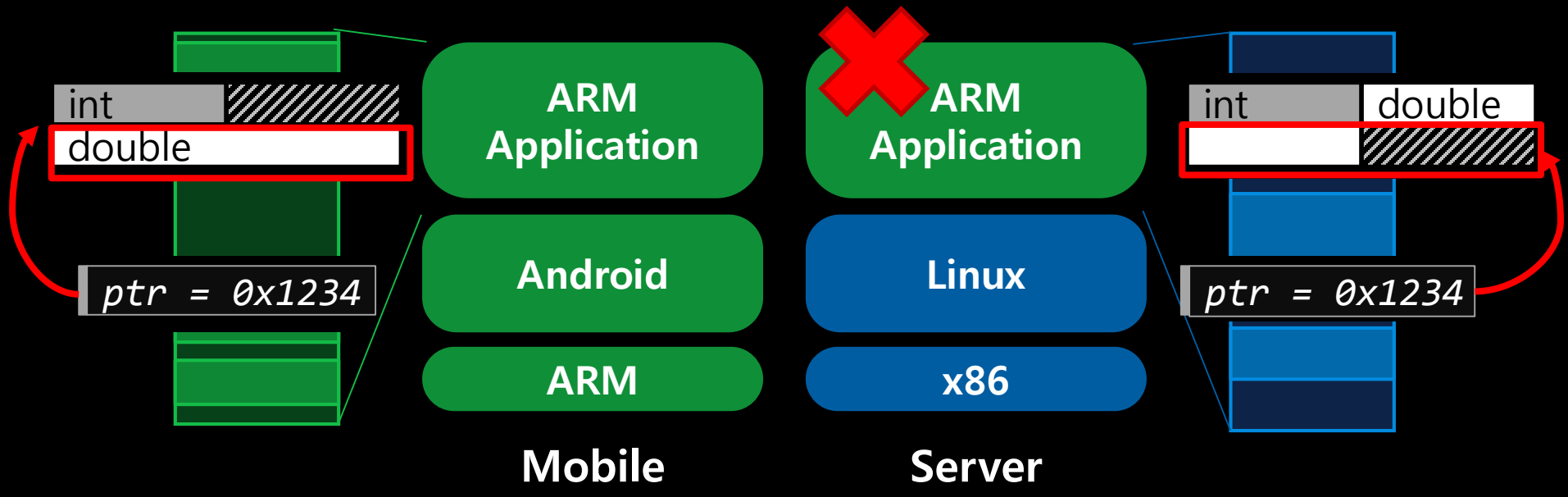


Different Architecture

Distinct Memory

Different Memory Layouts

Challenges in offloading native workloads

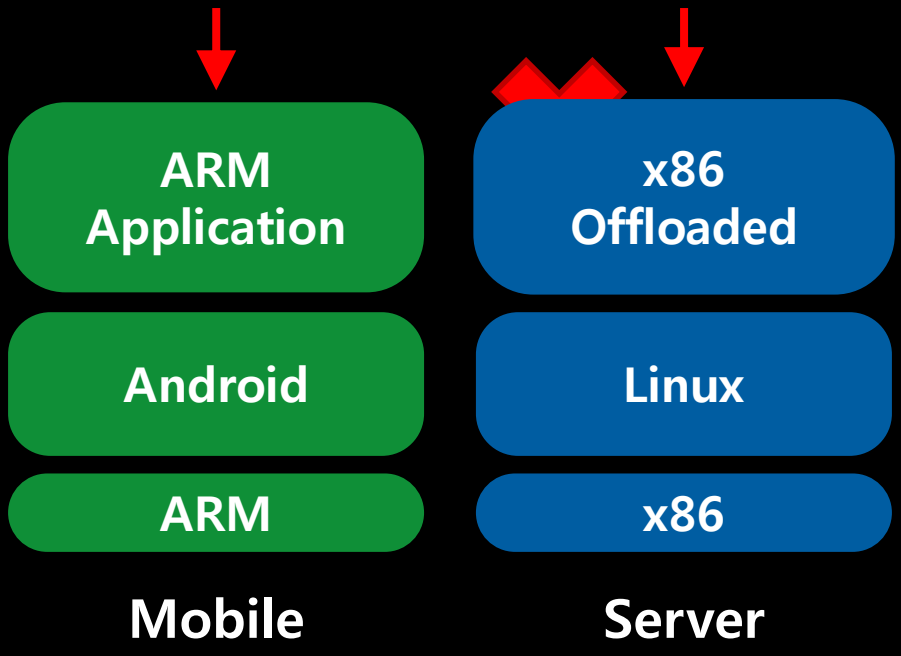


Different Architecture

Distinct Memory

Different Memory Layouts

Our Strategy 1: Compile Both Binaries!

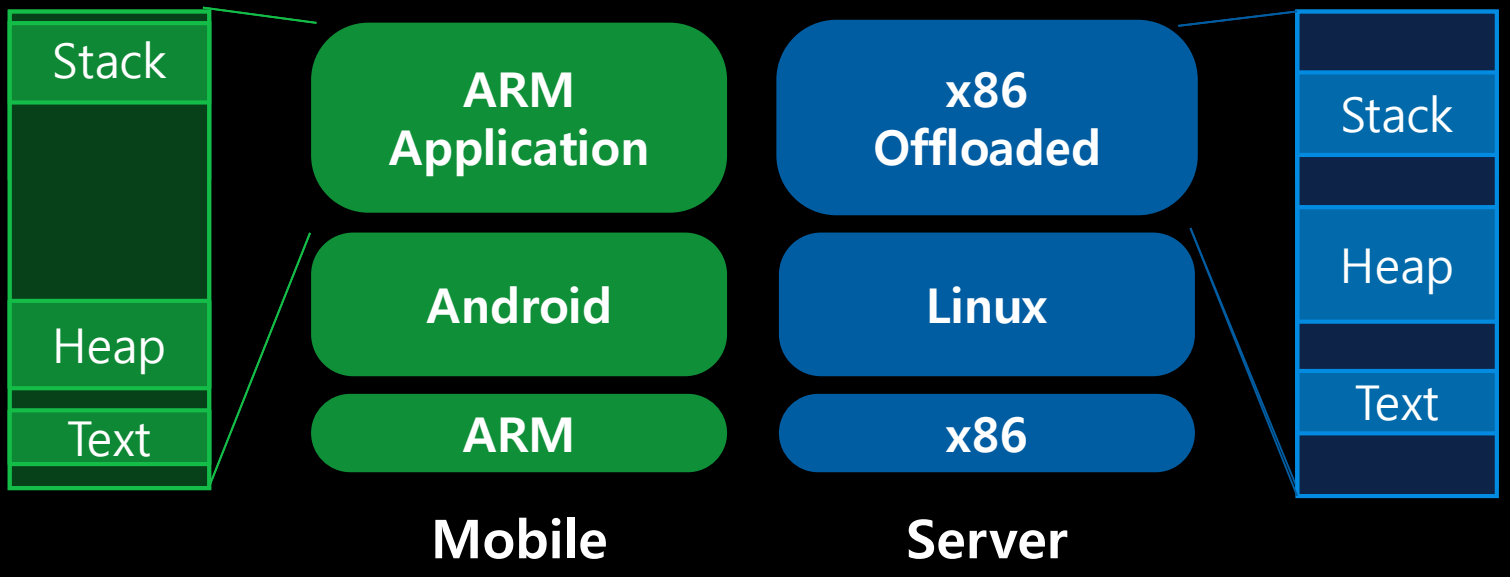


Different Architecture

Distinct Memory

Different Memory Layouts

Our Strategy 2: Unified Virtual Address

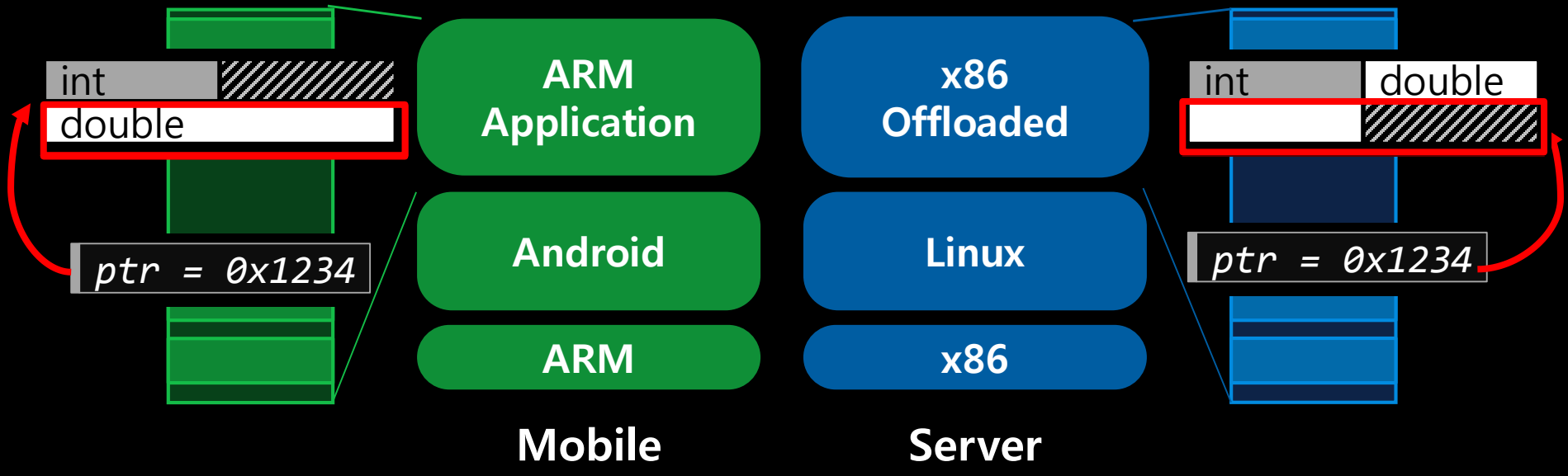


Different Architecture

Distinct Memory

Different Memory Layouts

Our Strategy 3: Unified Memory Layout

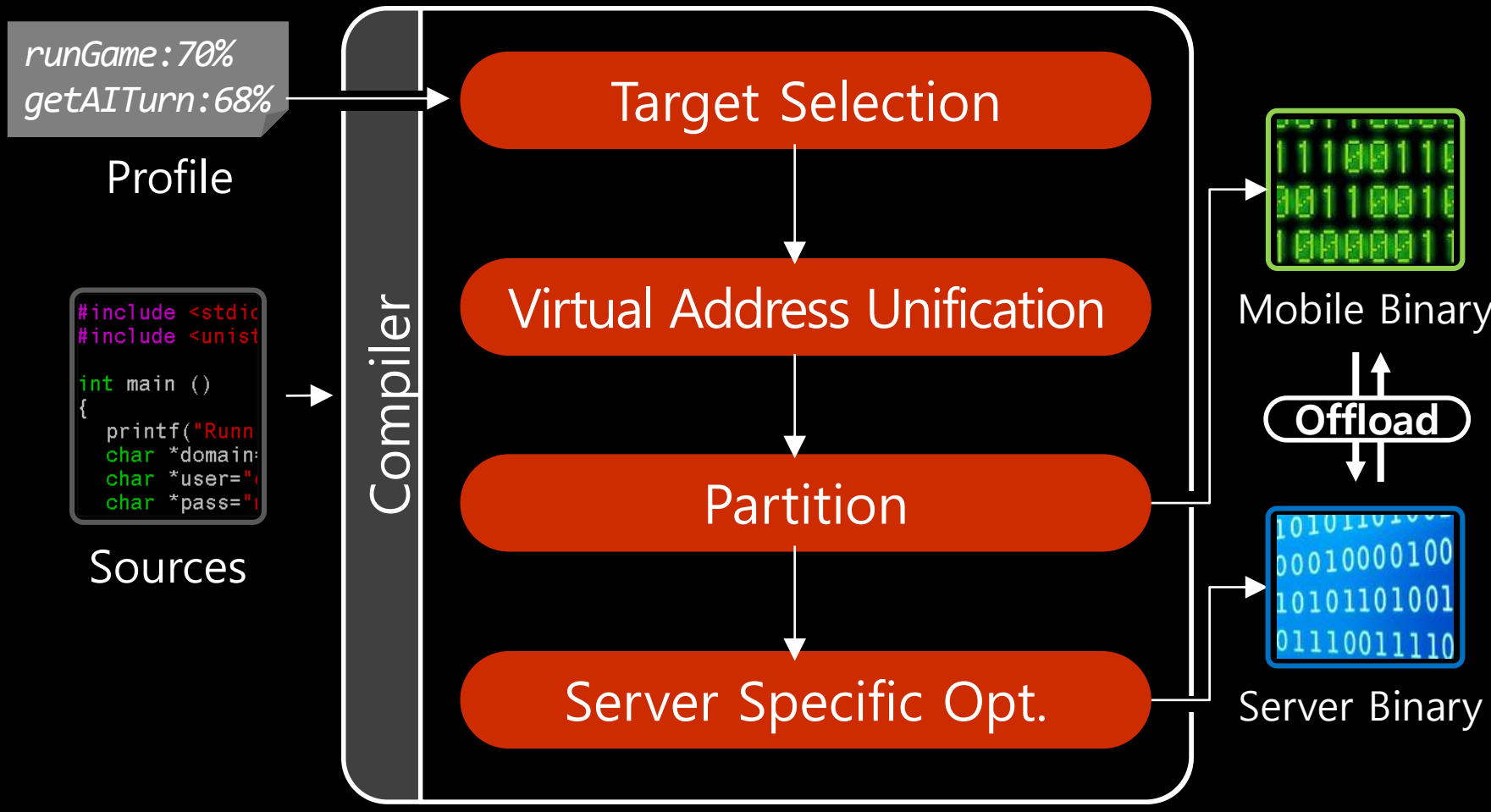


Different Architecture

Distinct Memory

Different Memory Layouts

Native Offloader : Structure Overview



Selecting Profitable Targets

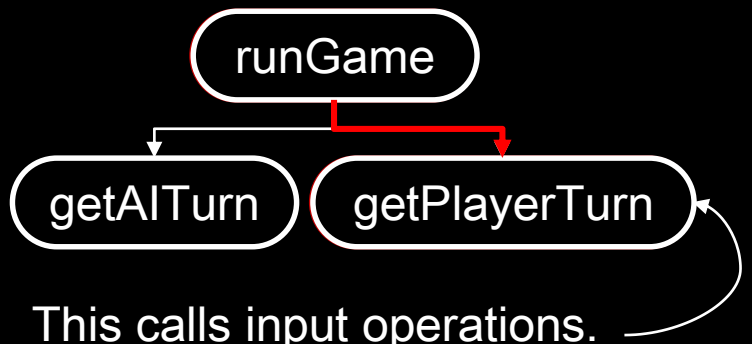
Candidate	Exec. Time	Call Count	Mem. Size
runGame	27 s	1	20 MB
getAITurn	26 s	3	12 MB
~~getPlayerTurn~~	~~1.5 s~~	~~3~~	~~10 MB~~

Estimation

Candidate	Ideal Gain	Comm.	Estimated Gain
runGame	**21.6 s**	**4 s**	**17.6 s**
getAITurn	**20.8 s**	**7.2 s**	**13.6 s**
~~getPlayerTurn~~	~~1.2 s~~	~~6 s~~	~~-4.8 s~~

Selecting Profitable Targets

Candidate	Exec. Time	Call Count	Mem. Size
~~runGame~~	~~27 s~~	~~1~~	~~20 MB~~
getAITurn	26 s	3	12 MB
~~getPlayerTurn~~	~~1.5 s~~	~~3~~	~~10 MB~~



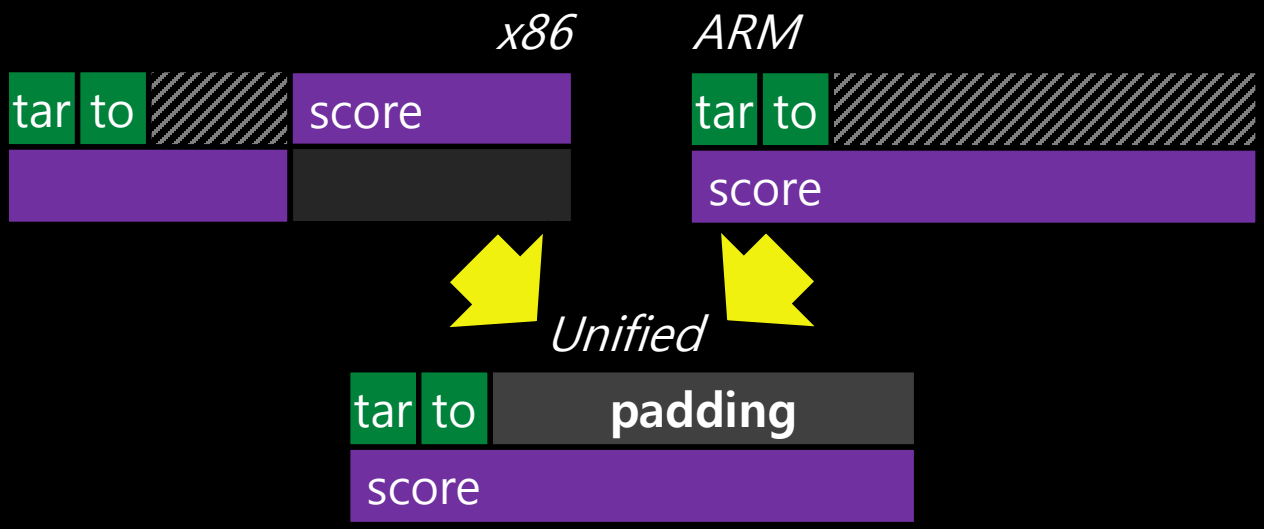
Estimation

Selected →

Candidate	Ideal Gain	Comm.	Estimated Gain
~~runGame~~	~~21.6 s~~	~~4 s~~	~~17.6 s~~
getAITurn	20.8 s	7.2 s	13.6 s
~~getPlayerTurn~~	~~1.2 s~~	~~6 s~~	~~-4.8 s~~

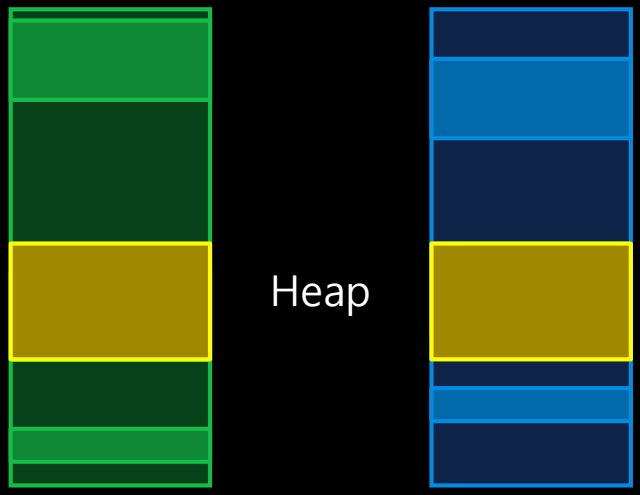
Unifying Structure Layouts

```
typedef struct {
  char tar, to; <pad>; double score;
} Move;
```



Unifying Heap Areas

```
board = u_malloc(sizeof(char) * 32);
u_free(board);
```

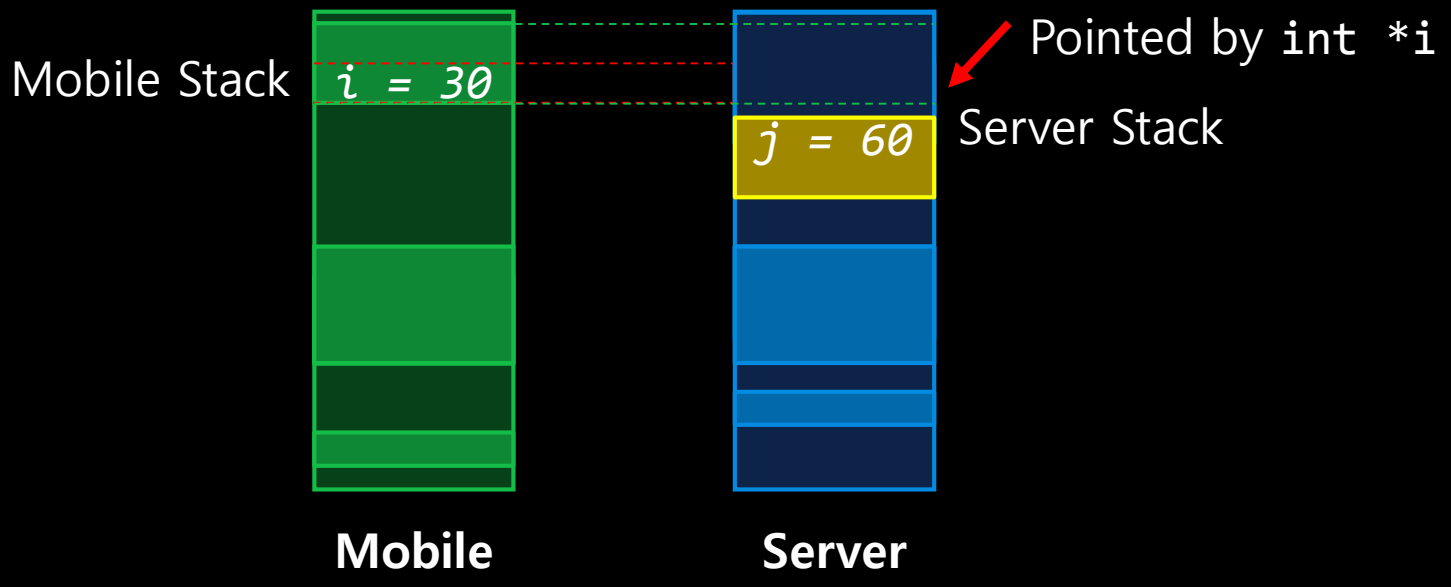


Aligning Two Stack Areas

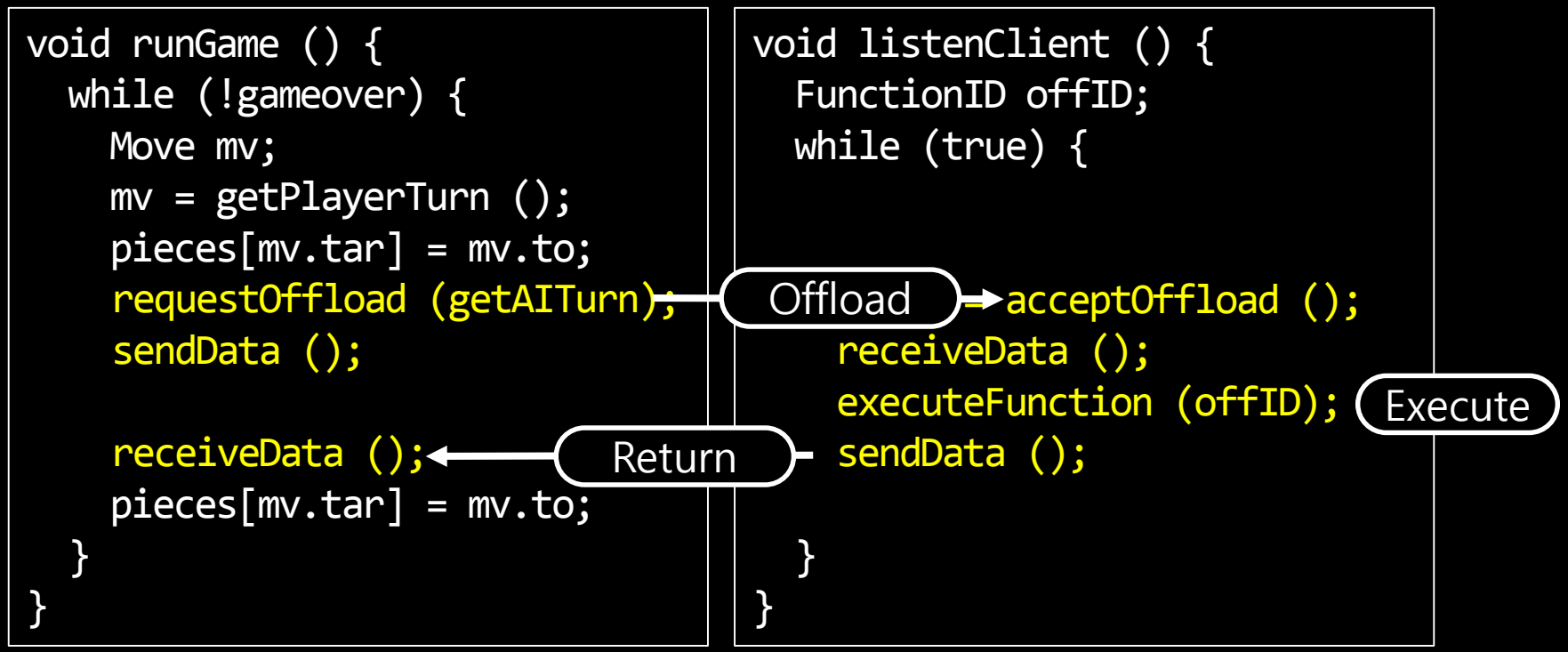
```
int foo () {
  int i = 30;
  bar (&i);
}
```

```
int bar (int *i) {
  int j = 60;
  if (*i!=30) crash;
}
```

*i==60
→ **Crash!**



Partitioning for The Separate Binaries



Server Specific Optimizations

Remote I/O

:: The server's request for I/O operations remotely.

Function Pointer Mapper

:: Maps the mobile's function address to the server's one

Please refer to our paper for more details!

Remaining Challenges for Mobile Apps

Multi-threaded Applications

:: Emerging mobile applications are multi-threaded.

Multi-language Support

:: Mobile apps are written in multiple languages.
ex) Android Apps w/ NDK (Java + C/C++)

So, this work uses SPEC benchmark suites.

Evaluation

Galaxy S5 as the Mobile Device, Dell XPS8700 as the Server

:: Mobile (2.5GHz Quad-core Krait 400)
Server (Intel 3.60GHz Quad-core i7-4790)

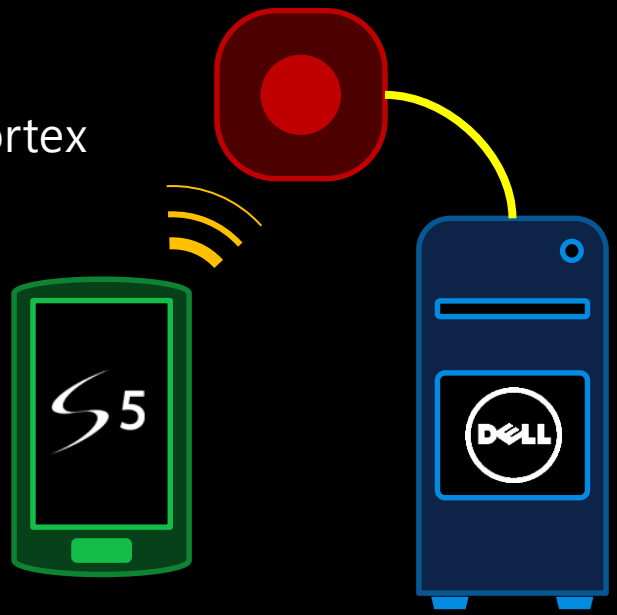
17 SPEC 2000/2006 C Benchmarks

:: LLVM Compile Error: 400.perlbench, 403.gcc
Non-profitable target: 197.parser, 254.gap, 255.vortex

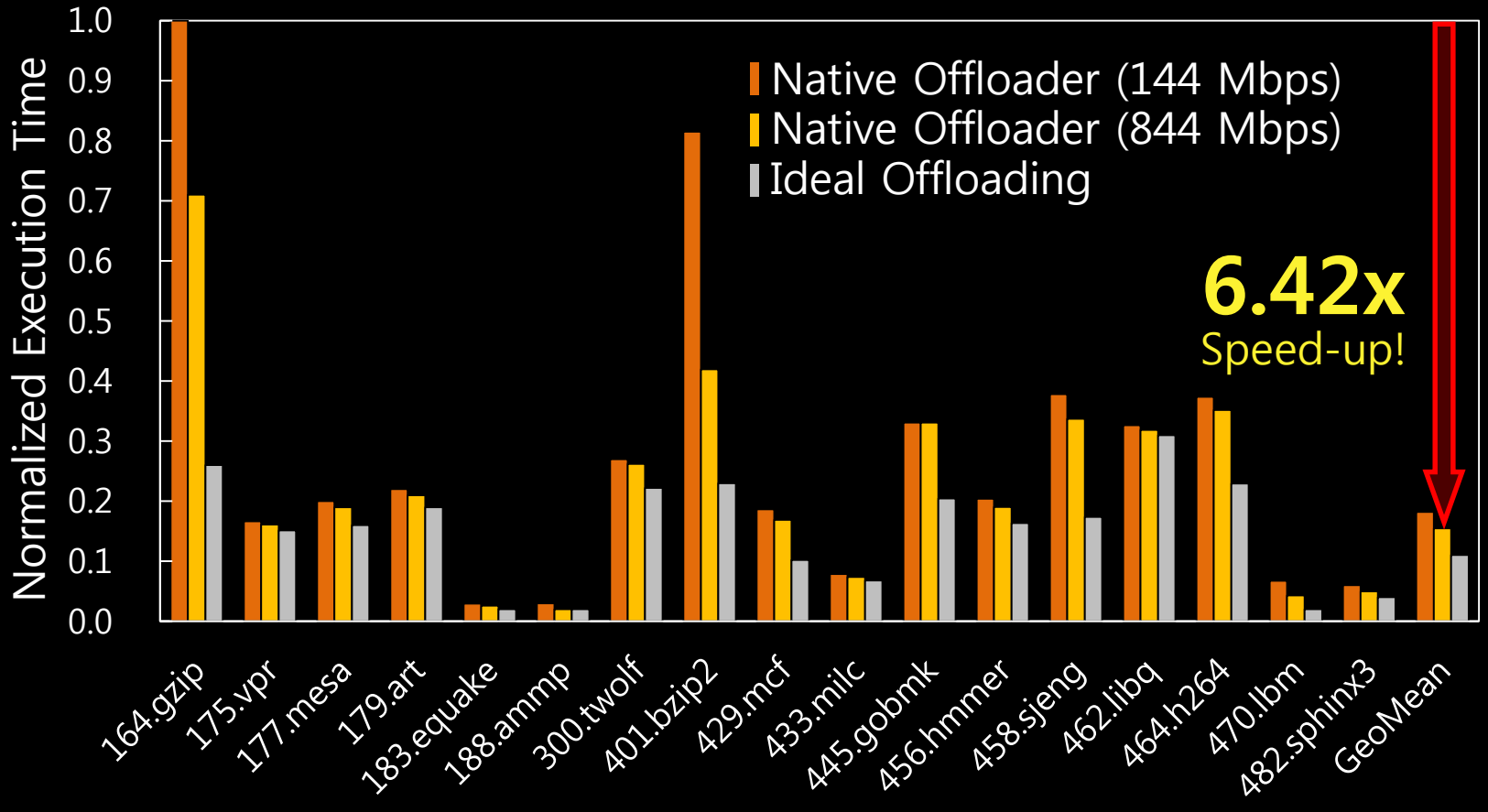
2 Different Network Bandwidths

:: 802.11n (Maximum 144Mbps)
802.11ac (Maximum 844Mbps)

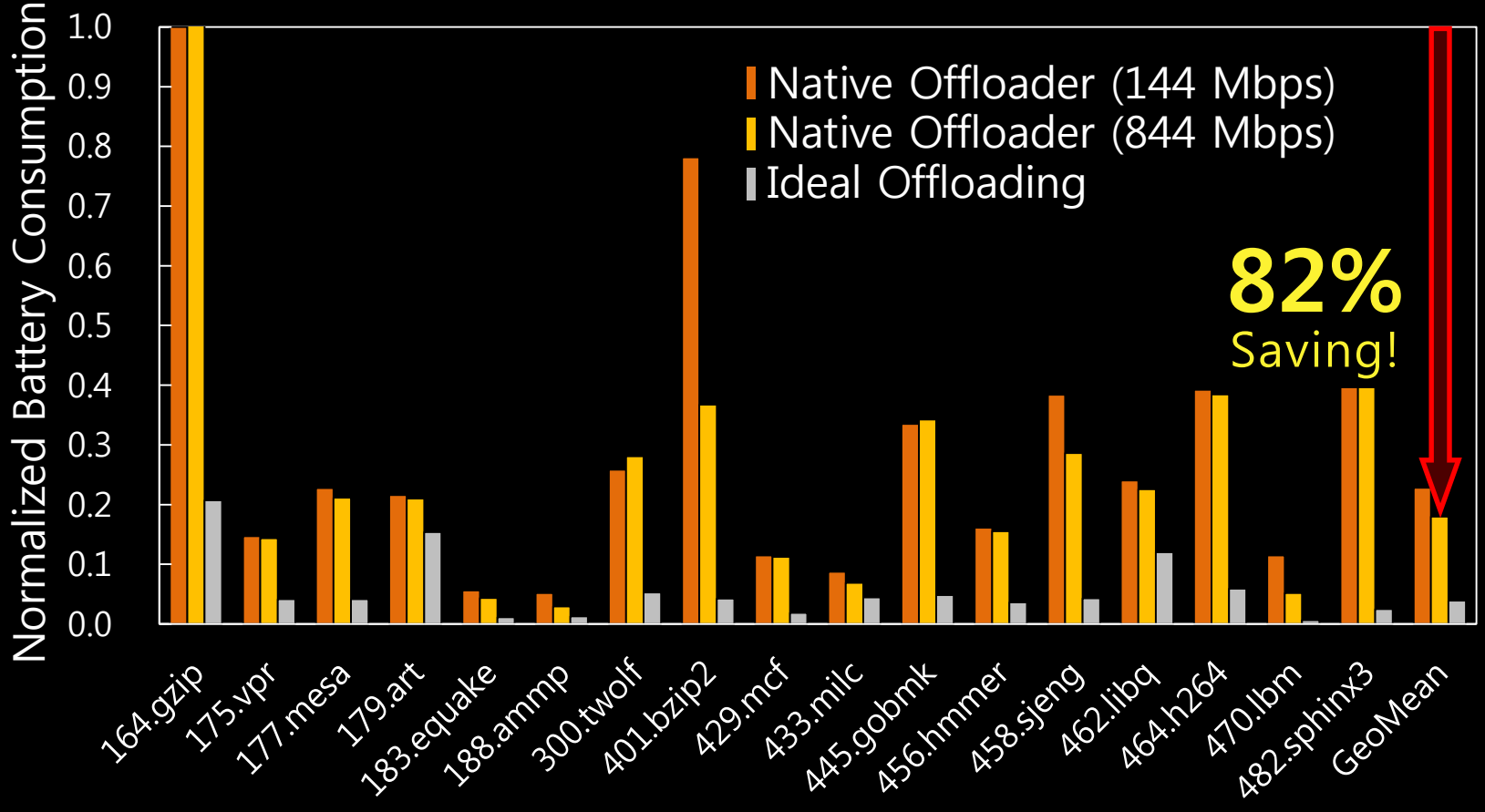
LLVM Compiler Framework



Normalized Execution Time



Normalized Battery Consumption



Conclusion

Native Offloader

:: Compiler/runtime cooperative offloading system for general-purpose native applications

:: To minimize offloading overheads, this work unified virtual address spaces.

Fast & Battery-friendly

:: 6.42x Speed-up / 82% Battery saving for 17 SPEC 2000/2006 C Benchmarks

The paper also includes remote I/O, function pointer mapping, pointer size / endianness translation, runtime system and comm. optimizations.

Image: http://www.worth1000.com/contests/27097/interspecies-teamwork-2

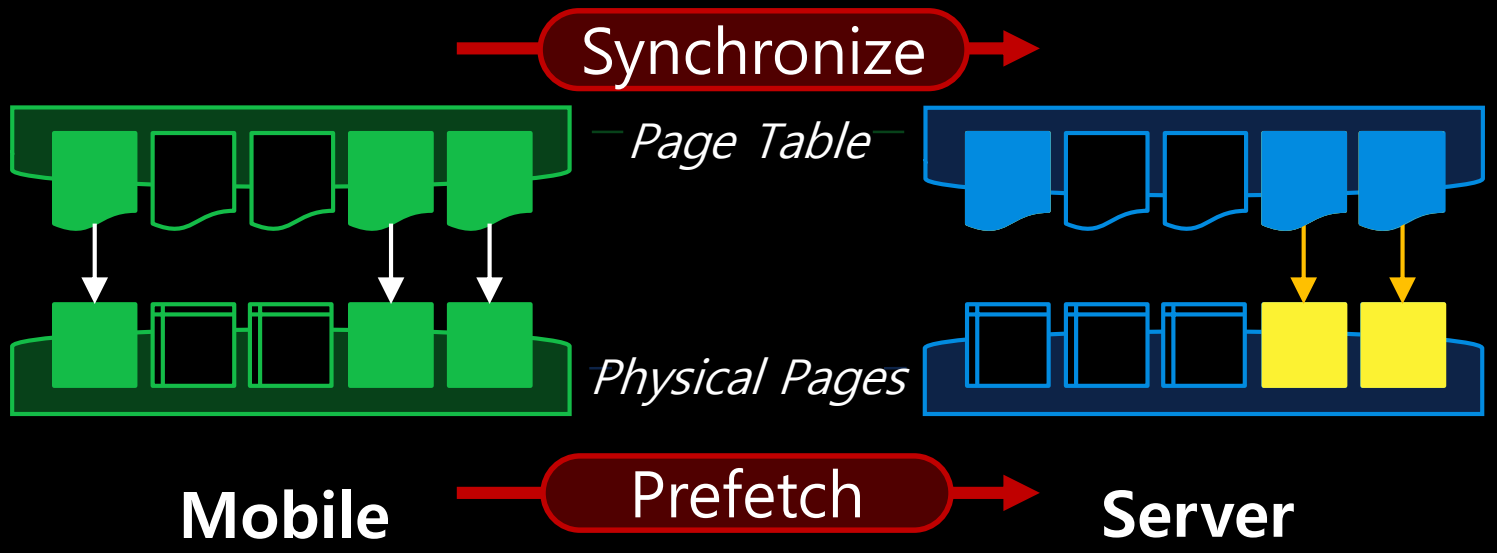
Native Offloader

Architecture-aware Automatic Computation Offload
for Native Applications

Backup Slides

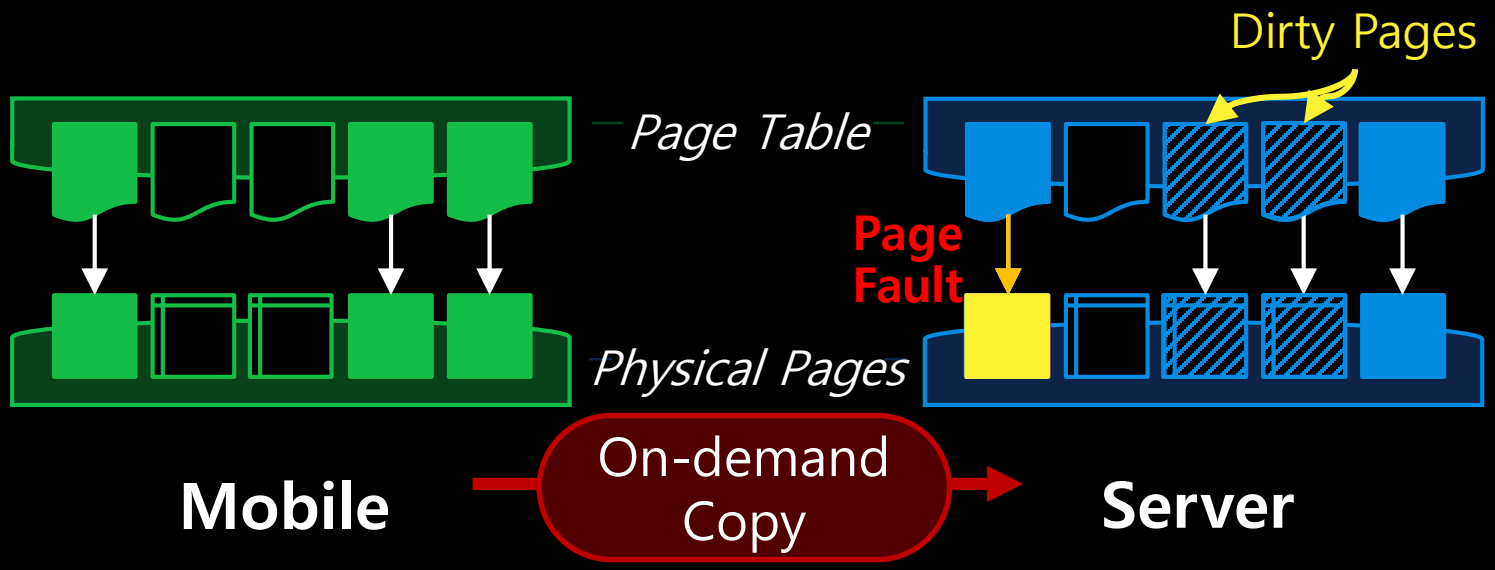
How It Works at Runtime

Stage: *Initialization*



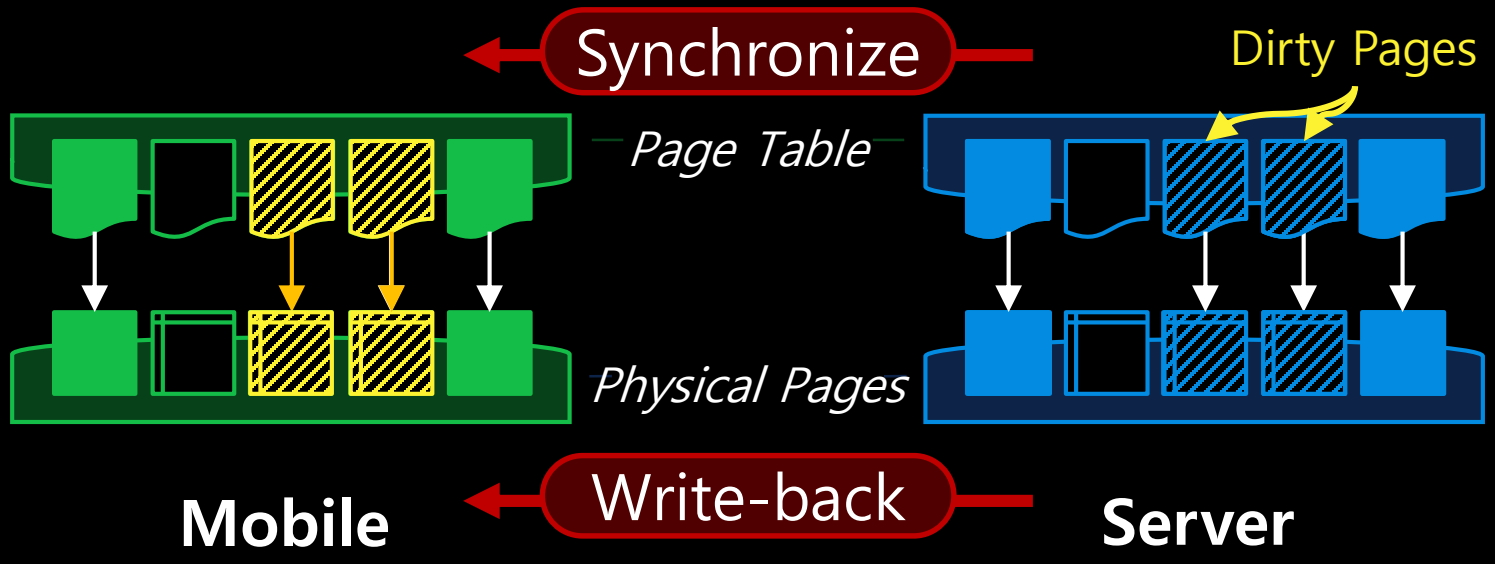
How It Works at Runtime

Stage: *Offloading Execution*

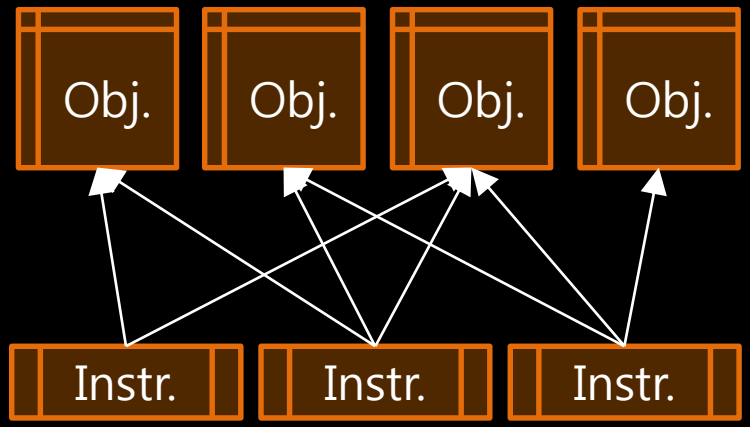


How It Works at Runtime

Stage: *Finalization*

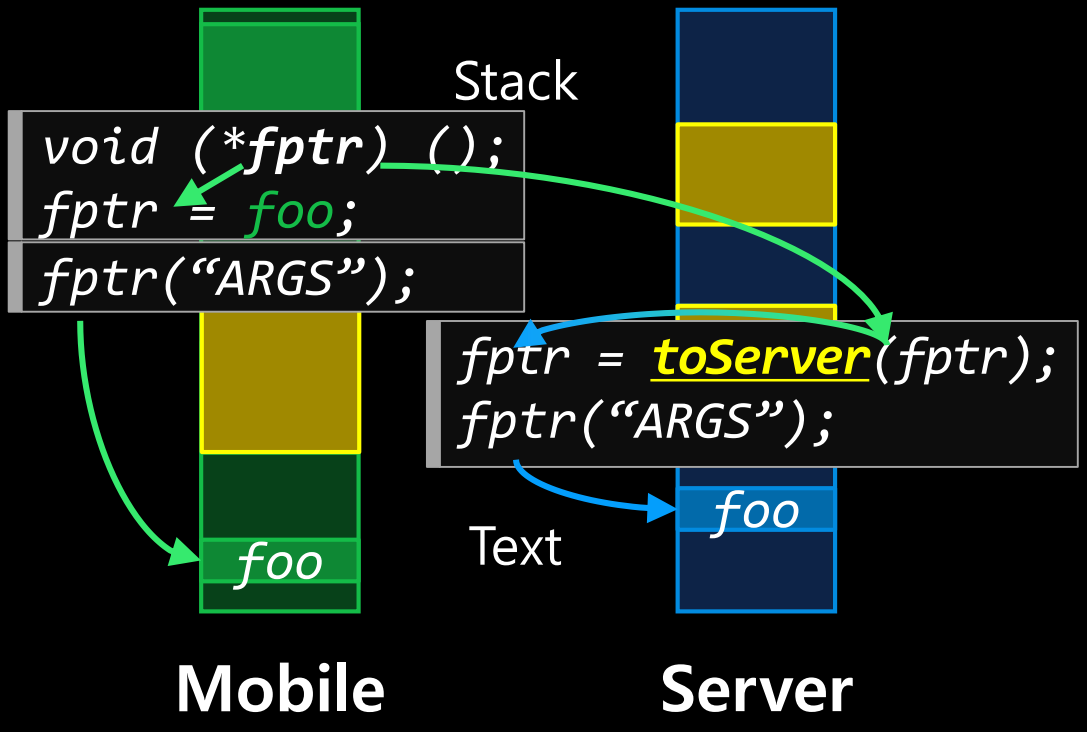


Several offloading systems are already proposed.



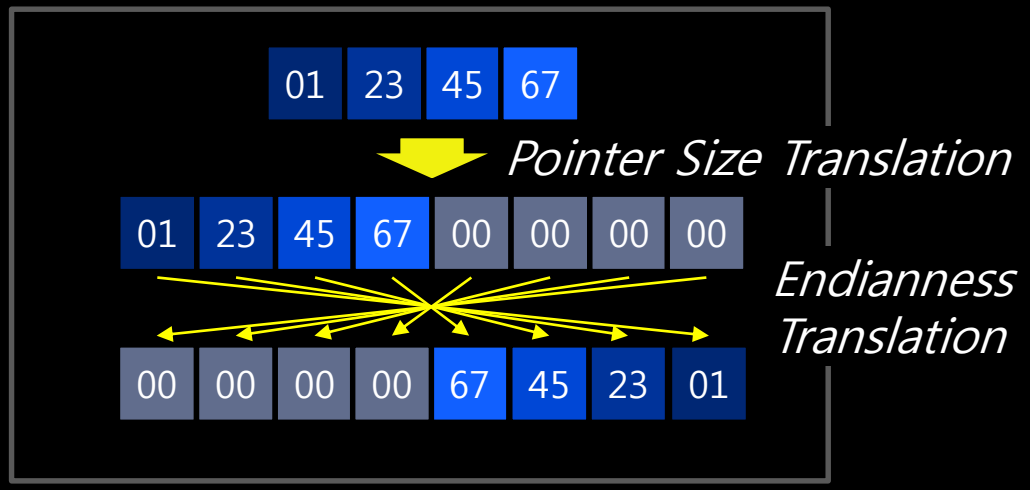
Pointer Analysis-based System
Li et al. (CASES`01), **Wang and Li** (PLDI`04)

Function Pointer Mapping



Unifying the Virtual Address Space

Installing Pointer Size & Endianness Translators



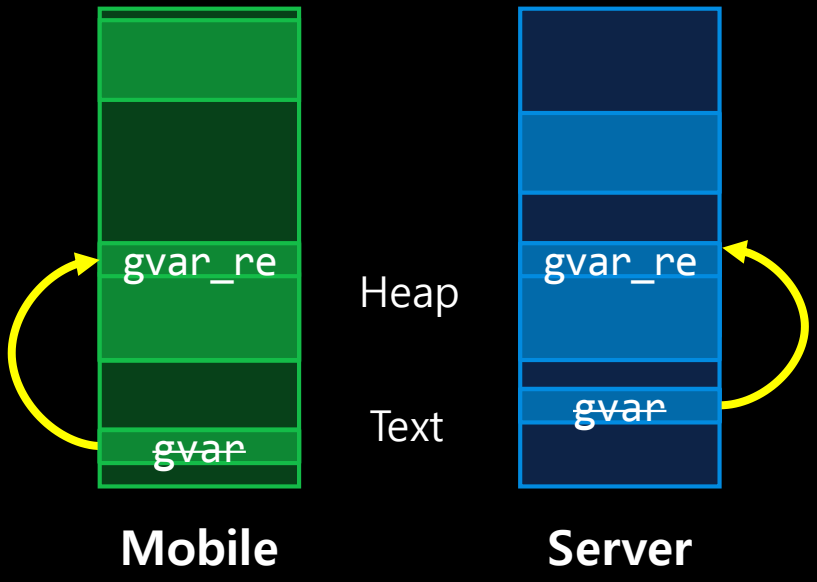

```
int value = *ptr32;
```

```
    ptr64 = zext (ptr32);
int value = toLittle(*ptr64);
```

Reallocating Global Variables

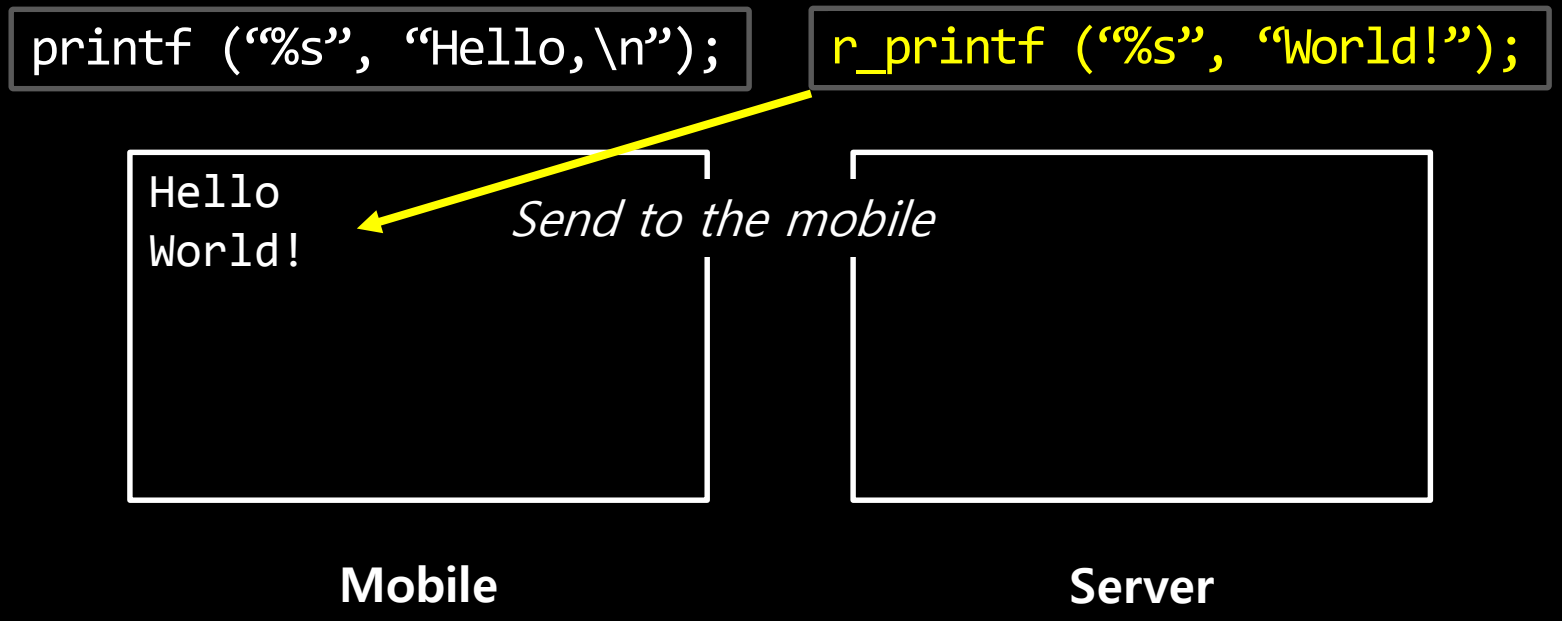
```
int *gvar_re= u_malloc(4);
int *gptr = gvar_re;
```

```
int foo () {
  *gptr = 400;
}
```

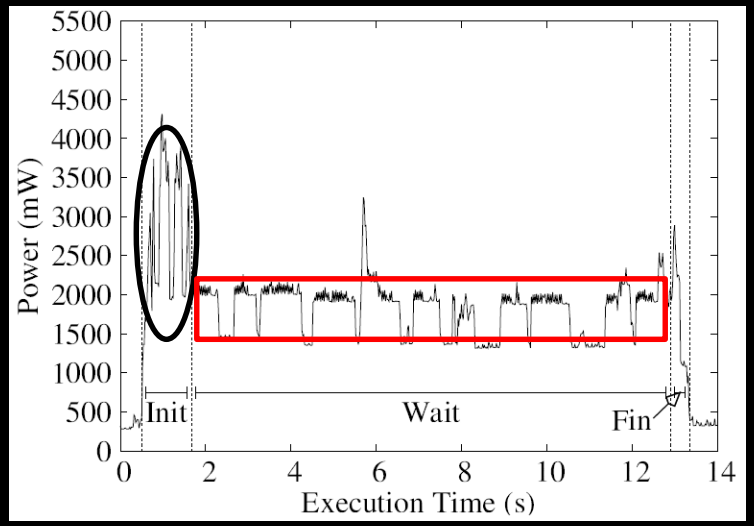


Server Specific Optimizations

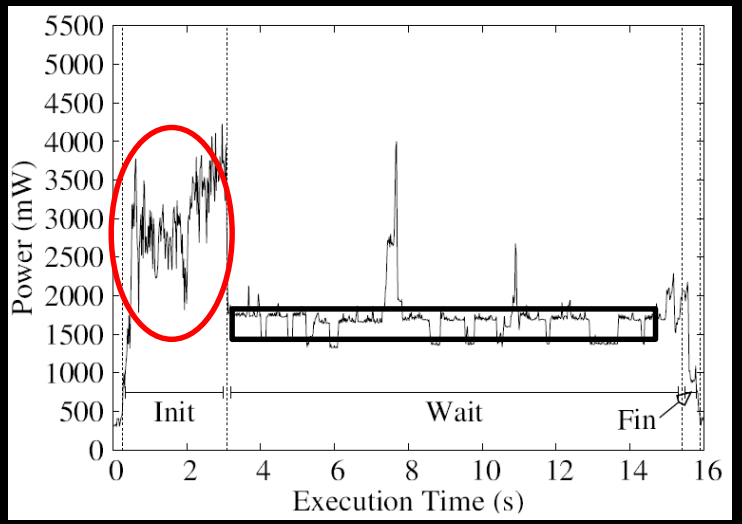
Installing Remote I/O APIs



Runtime Battery Consumption Pattern



445.gobmk (Fast, 802.11ac)



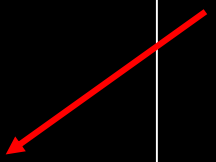
445.gobmk (Slow, 802.11n)

Initial Prefetching Overhead : 802.11ac < 802.11n
\+ Total Remote I/O Overhead : 802.11ac ≫ 802.11n

Total Battery Overhead : 802.11ac > 802.11n

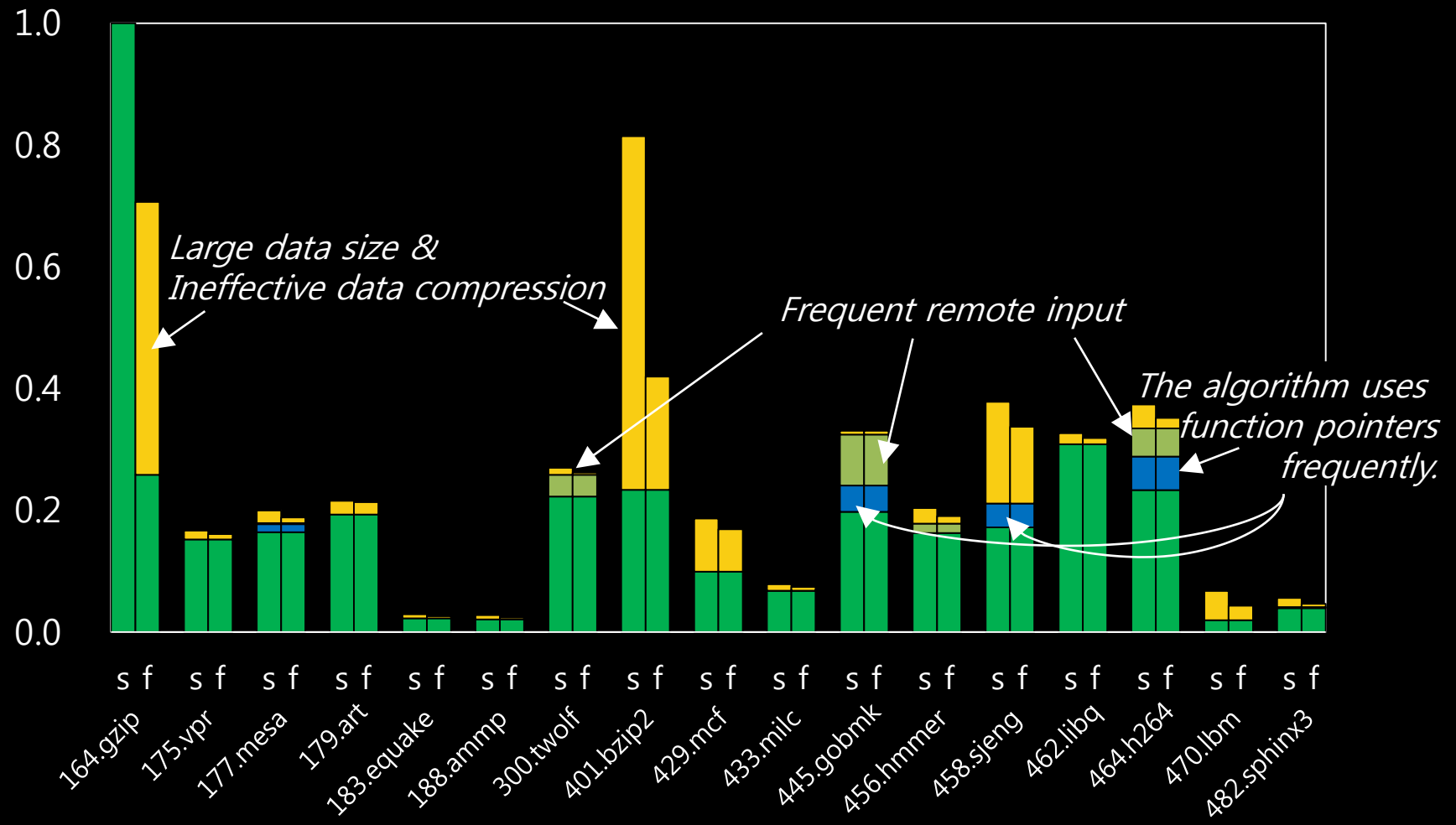
Runtime Estimator

```
void runGame () {
  while (!gameover) {
    Move mv;
    mv = getPlayerTurn ();
    pieces[mv.tar] = mv.to;
    if (profitable (getAITurn)) {
      requestOffload (getAITurn);
      sendData ();
      receiveData ();
    }
    pieces[mv.tar] = mv.to;
  }
}
```



Real-time bandwidth information

Overhead Analysis



Gwangmu Lee
iss300@postech.ac.kr

Compiler Research Lab (CORELAB)
Dept. of Computer Science & Engineering
POSTECH, Korea

Thank you for your attention!

QUESTIONS & ANSWERS