Embedded Software Streaming via Block Stream

A Dissertation

by

Pramote Kucharoen

Dissertation Advisor

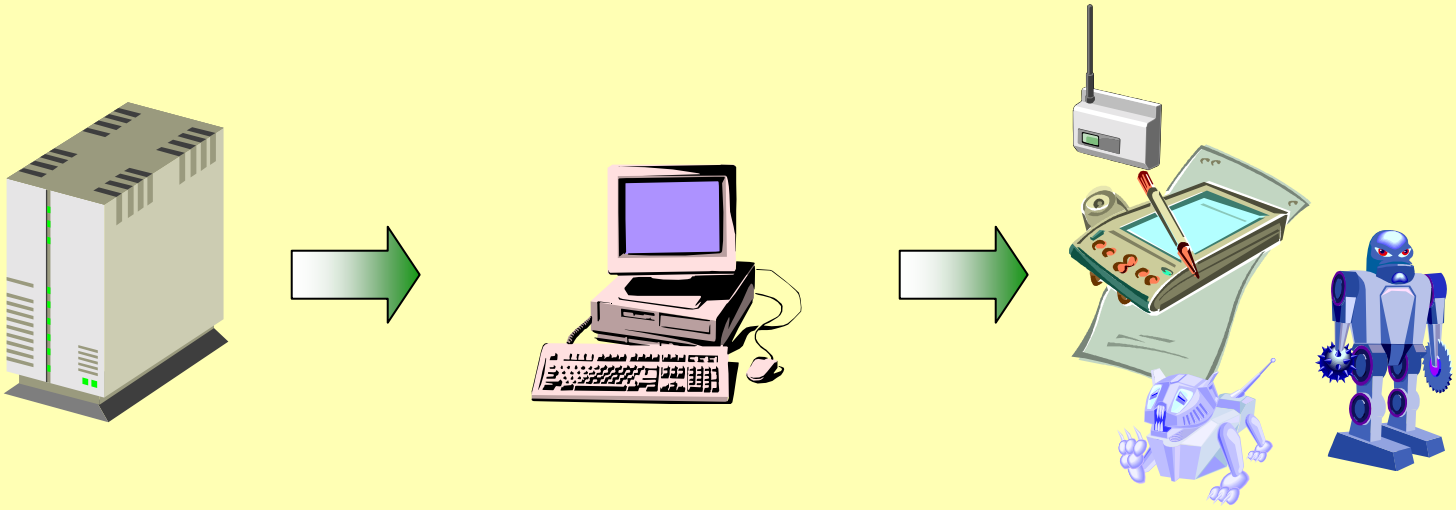
Professor Vincent J. Mooney III

7 April 2004

Outline

- Introduction
- Related Work
- Block Streaming
- Stream-Enabled Program Files
- Stream-Enabled File I/O
- Performance Enhancement
- Experiments and Results
- Conclusion

Introduction

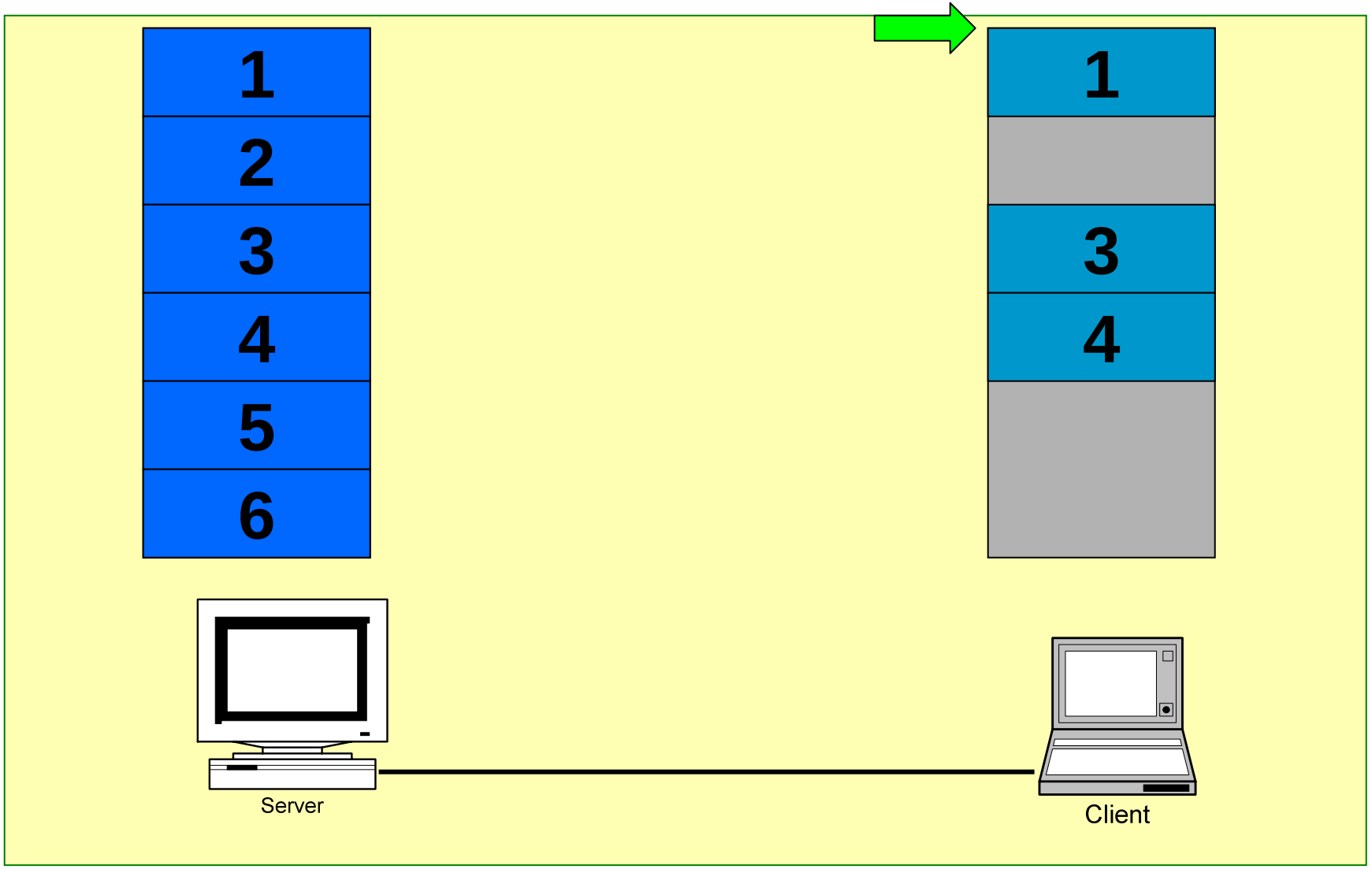


Package → Download & Remote execution → Software Streaming

Running Applications

- On clients (download and install)
  - Advantages
    - Less servers' computing requirements
    - Faster user interaction after installation complete
  - Disadvantages
    - Long download time
    - Clients' limited resources
    - Unused features downloaded
- On servers (remote execution)
  - Advantages
    - Applications not downloaded
    - Less clients' computing requirements
  - Disadvantages
    - Overloading of servers
    - Slower user interactions
    - Connection loss

Software Streaming



Definitions

- **Software streaming**
  - **A method for allowing the execution of stream-enabled software even while transmission/streaming of the software may still be in progress**
- **Application load time**
  - **the amount of time from when the application is selected for download to when the application can be executed**
- **Application suspension time**
  - **the amount of time from when the application is suspended due to missing code to when the application can be resumed**

Outline

- Introduction
- **Related Work**
- Block Streaming
- Stream-Enabled Program Files
- Stream-Enabled File I/O
- Performance Enhancement
- Experiments and Results
- Conclusion

Related Work (1)

- **Java**
  - **Allows execution without obtaining all classes**
  - **Sends class files when requested**
  - **Suspends the application for every class not in memory**
  - **Requires JVM**
  - **Assumes clients has enough memory to store the entire application**

Related Work (2)

- **Software caching [CH02]**
  - **Has high occurrence of application suspension**
- **Liquid software [JH96]**
  - **Requires a fast (“gigabit”) compiler**

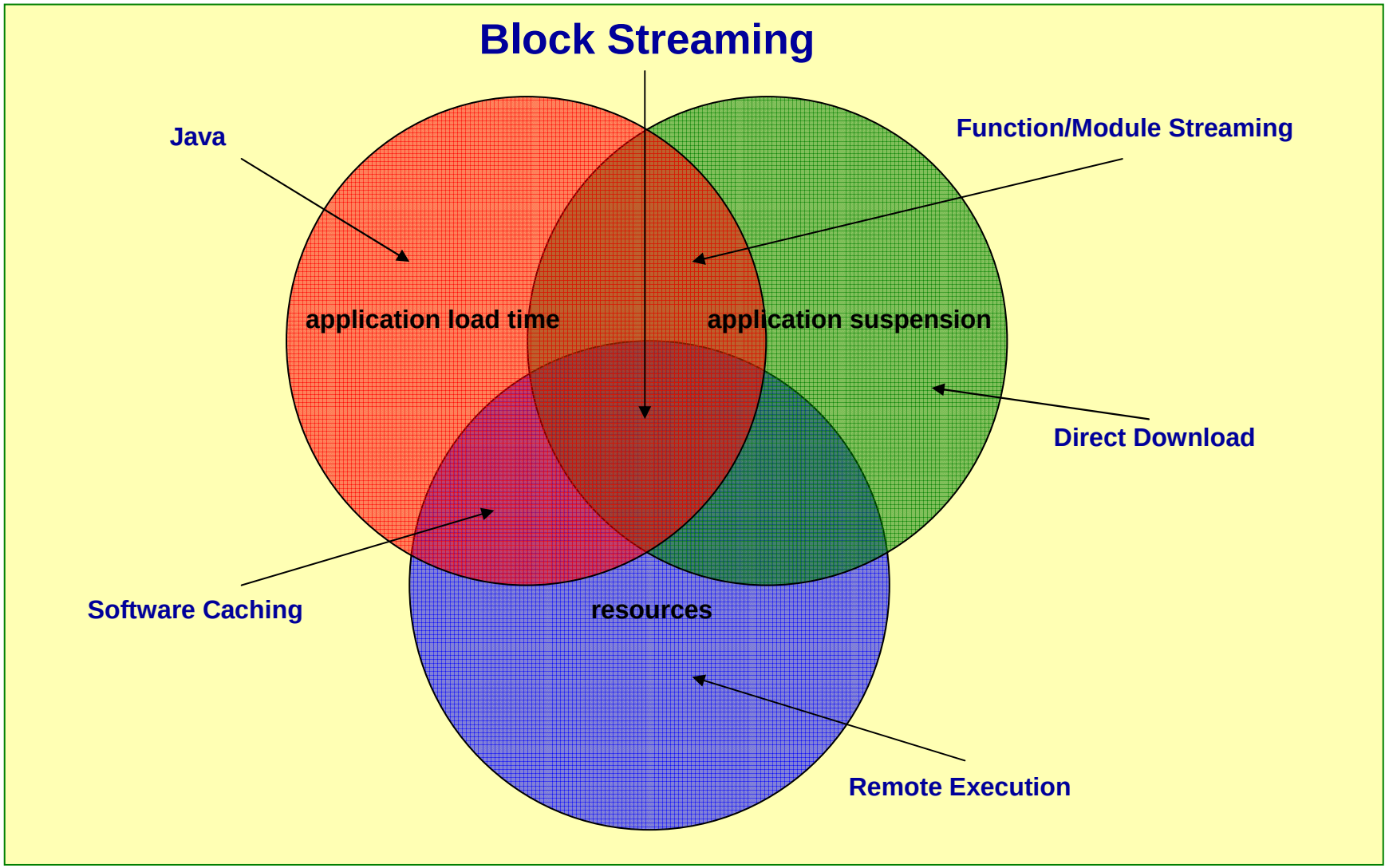
Related Work (3)

- **Function/module streaming [CK98][UR01]**
  - **Allows execution without obtaining all functions**
  - **Transfers functions speculatively to reduce the occurrence of application suspension**
  - **Assumes clients has enough memory to store the entire application**
  - **More difficult to manage memory**

Related Work (4)

- **Block streaming**
  - **Divides files into blocks**
  - **Streams at block level**
  - **No known prior work**

Related Work (4)



Outline

- Introduction
- Related Work
- **Block streaming**
- Stream-Enabled Program Files
- Stream-Enabled File I/O
- Performance Enhancement
- Experiments and Results
- Conclusion

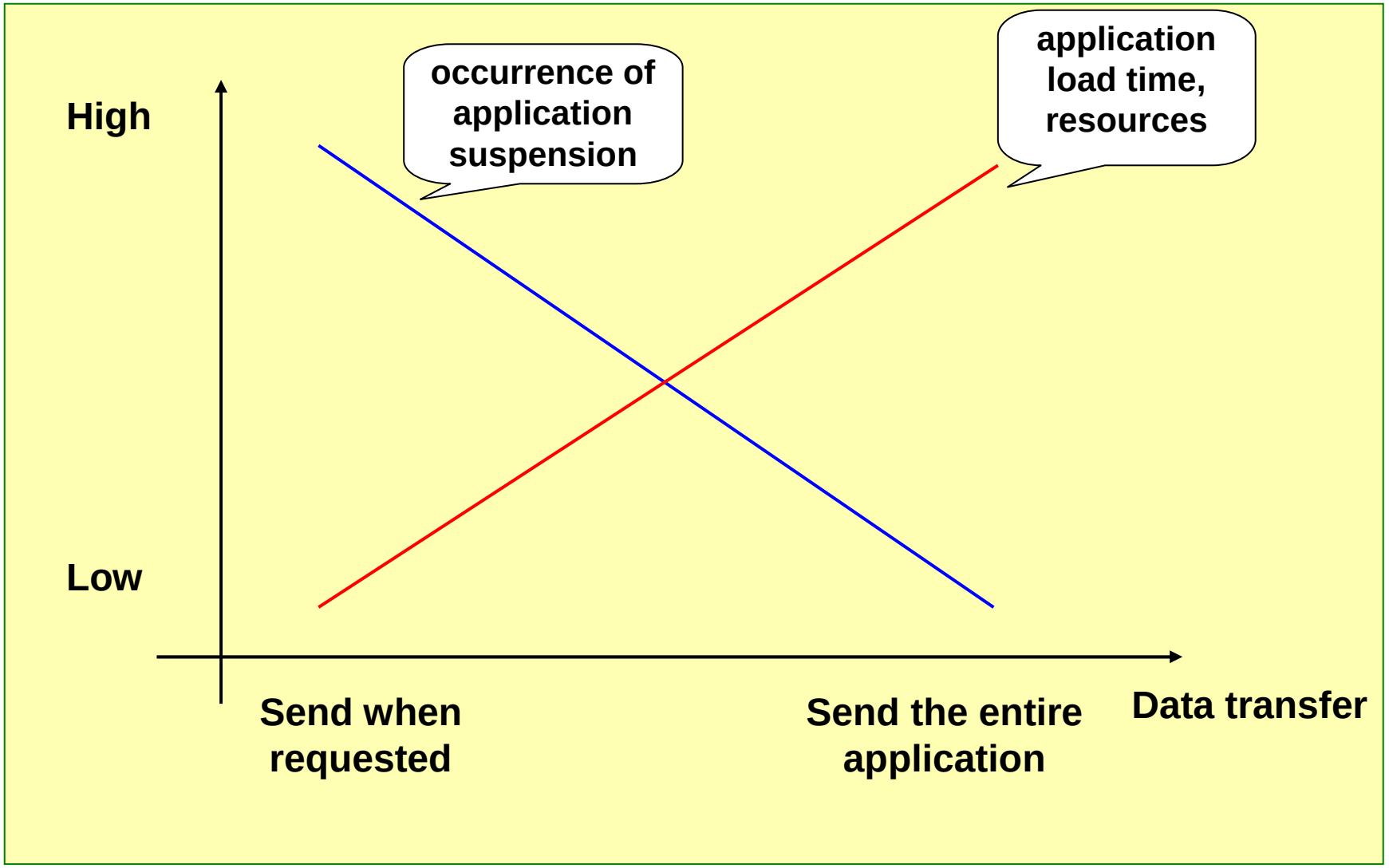
Block Streaming

- **Does not require virtual machines, virtual file system, compiler at client, special support from OS**
- **Uses a binary rewriting technique**
- **Supports embedded applications and small memory footprint devices**
- **Implements at user level (not OS dependant)**
- **Provides stream-enabled file I/O support**

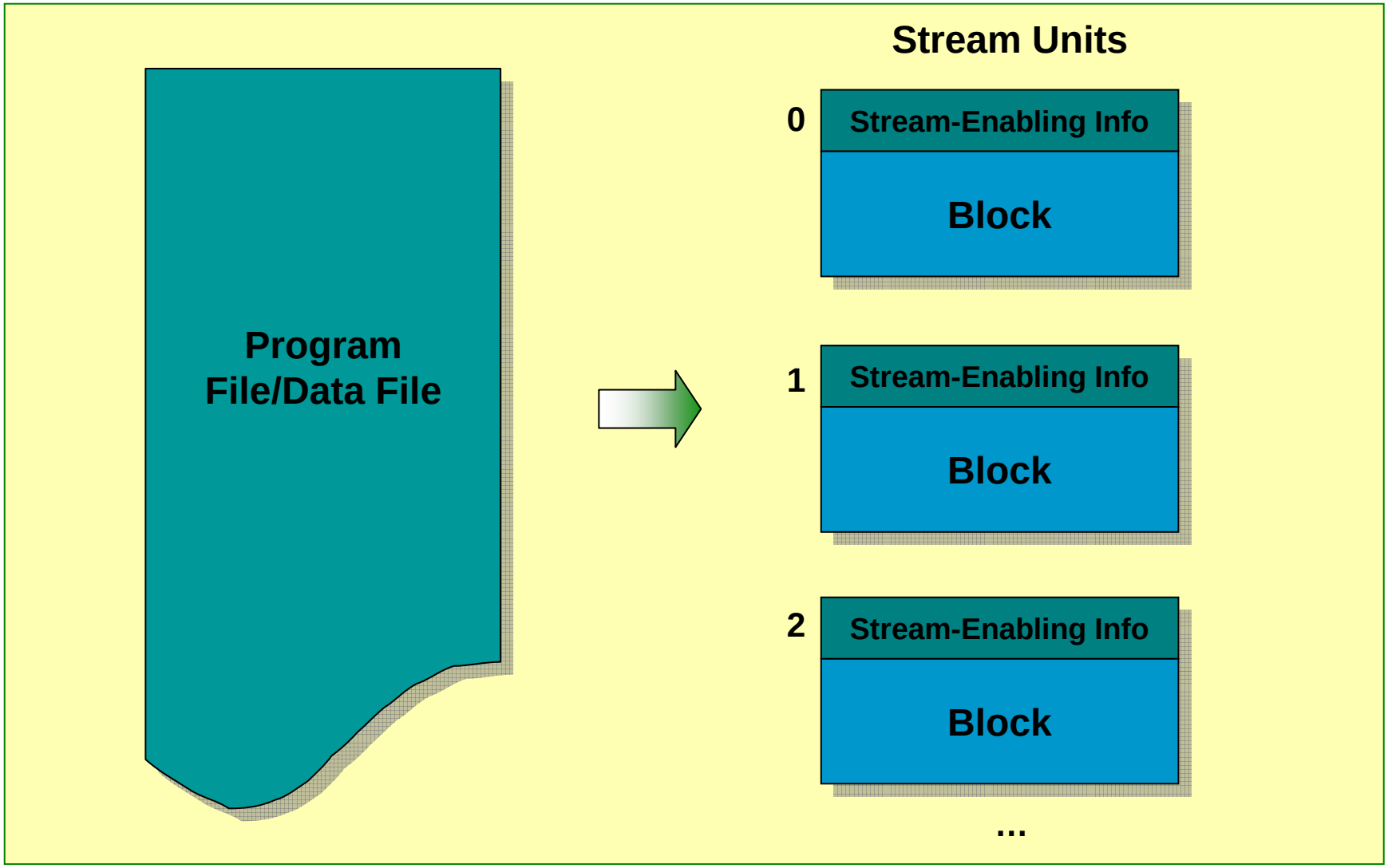
Objectives

- To reduce application load time
- To reduce application suspension time and occurrence of application suspension due to missing code/data
- To efficiently utilize resources such as bandwidth and memory
- To support small memory footprint embedded devices
- To optimize the above four objectives simultaneously (as opposed to tradeoffs)

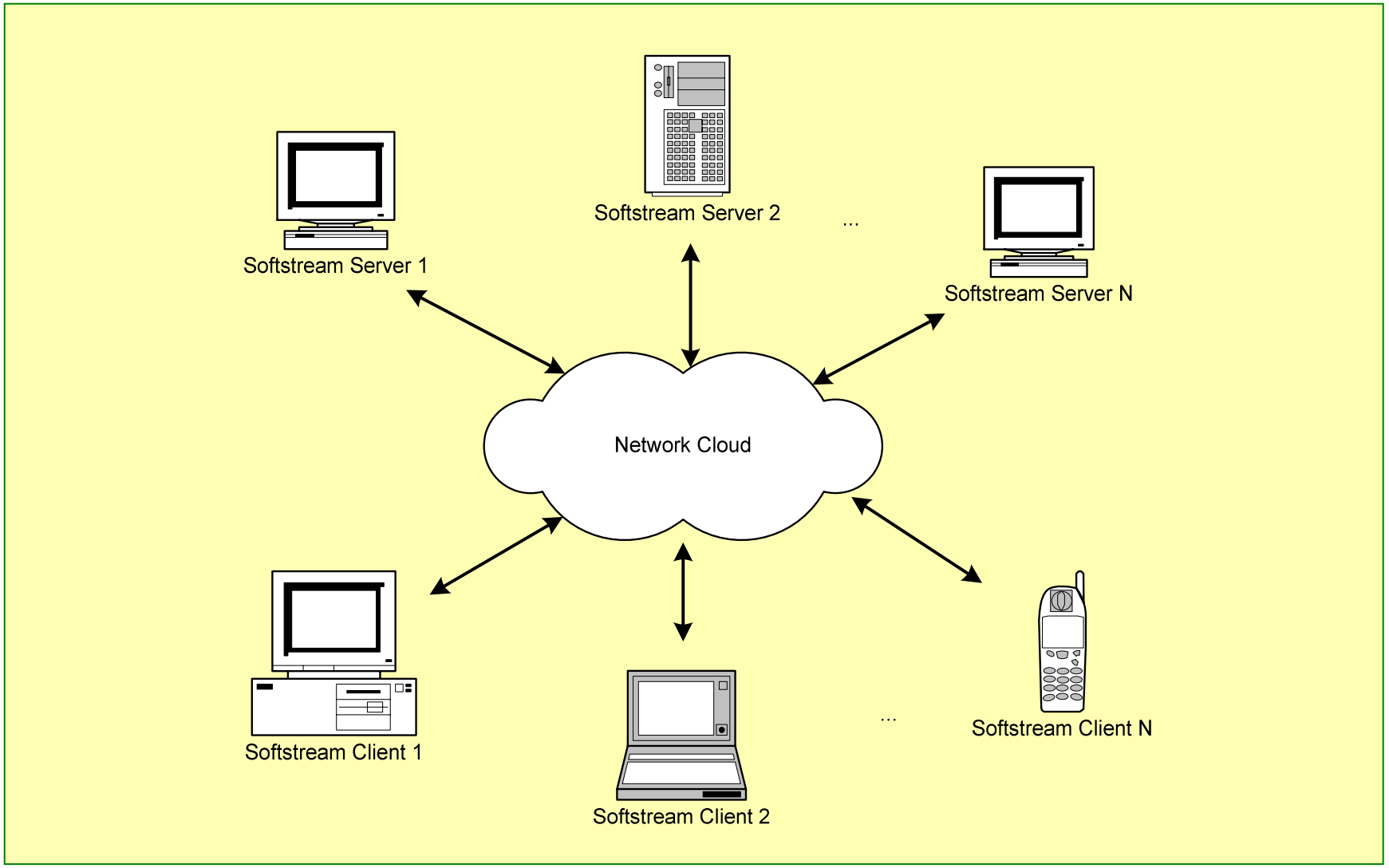
Objective tradeoffs



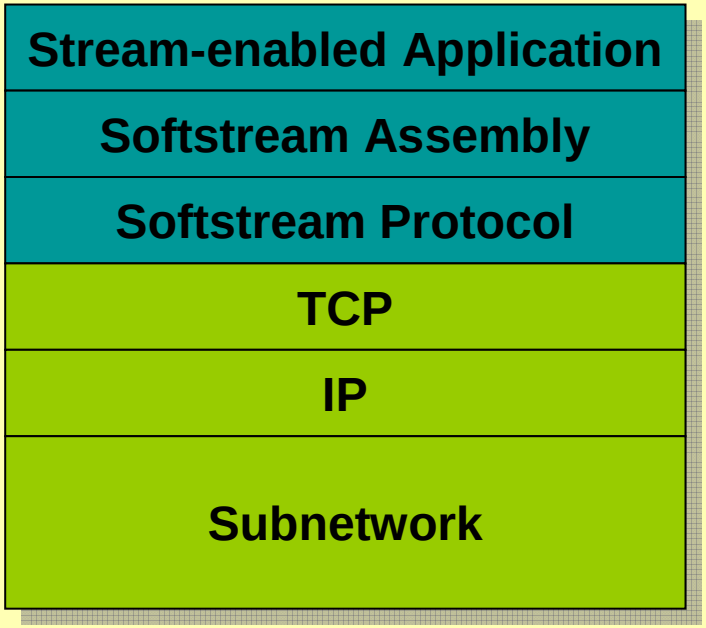
Block Streaming: Stream Units



Softstream Client/Server Model



Softstream Protocol Layers

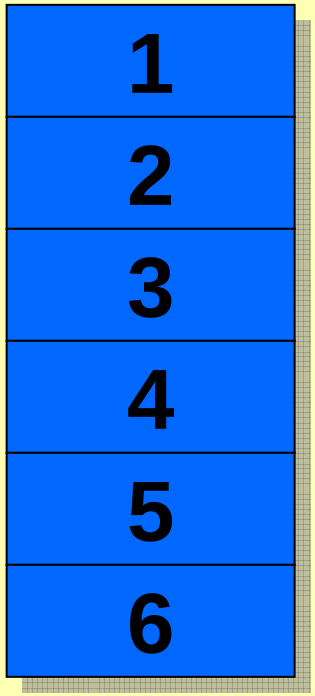


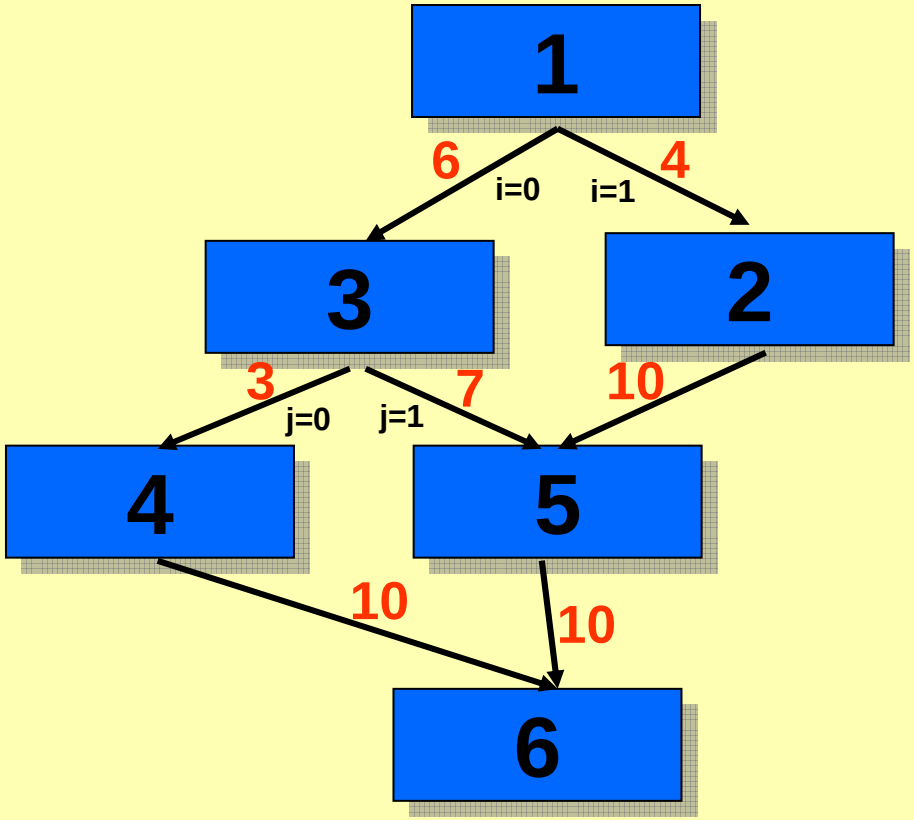
OSI Layers:

- Application
- Presentation
- Session
- Transport
- Network
- Link
- Physical

Transmission Profile

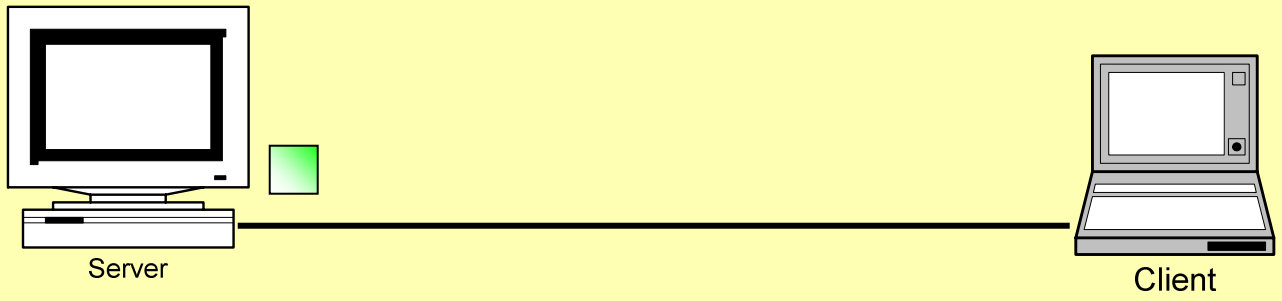
- Is created using a profiling method
- Is used to send stream units





Flow Control (1)

- **Continuous stream**
  - **Sends stream units according to the transmission profile**
  - **Restarts a new sequence when requesting a missing stream unit**



Flow Control (2)

- **On-demand stream**
  - **Sends stream units according to the transmission profile based on resource constraints**

Performance Metrics

- Overhead
  - Bandwidth
  - Memory
  - Processing time
- Application load time
- Application suspension time
- Occurrence of application suspensions due to missing code/data

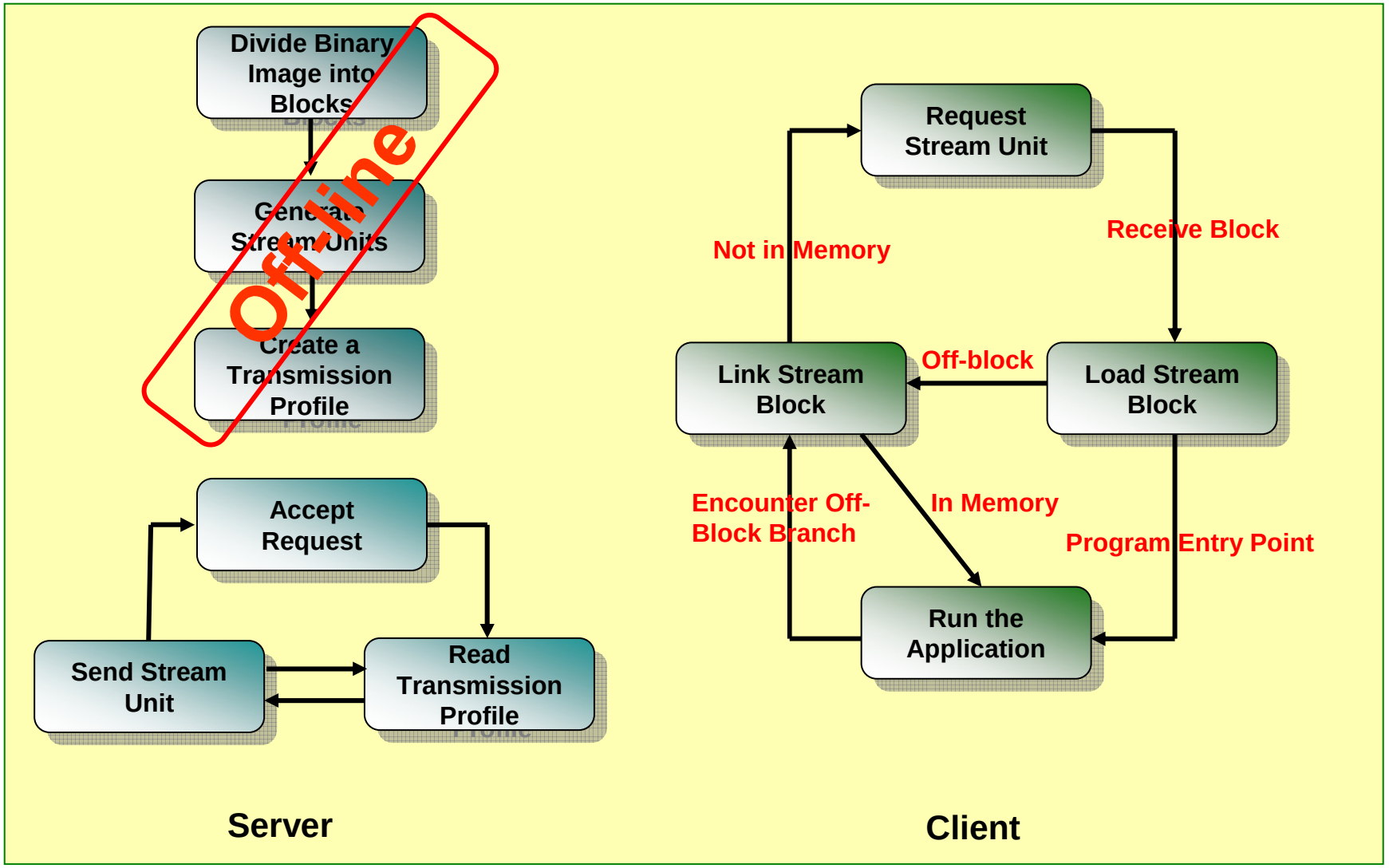
Outline

- Introduction
- Related Work
- Block Streaming
- **Stream-Enabled Program Files**
- Stream-Enabled File I/O
- Performance Enhancement
- Experiments and Results
- Conclusion

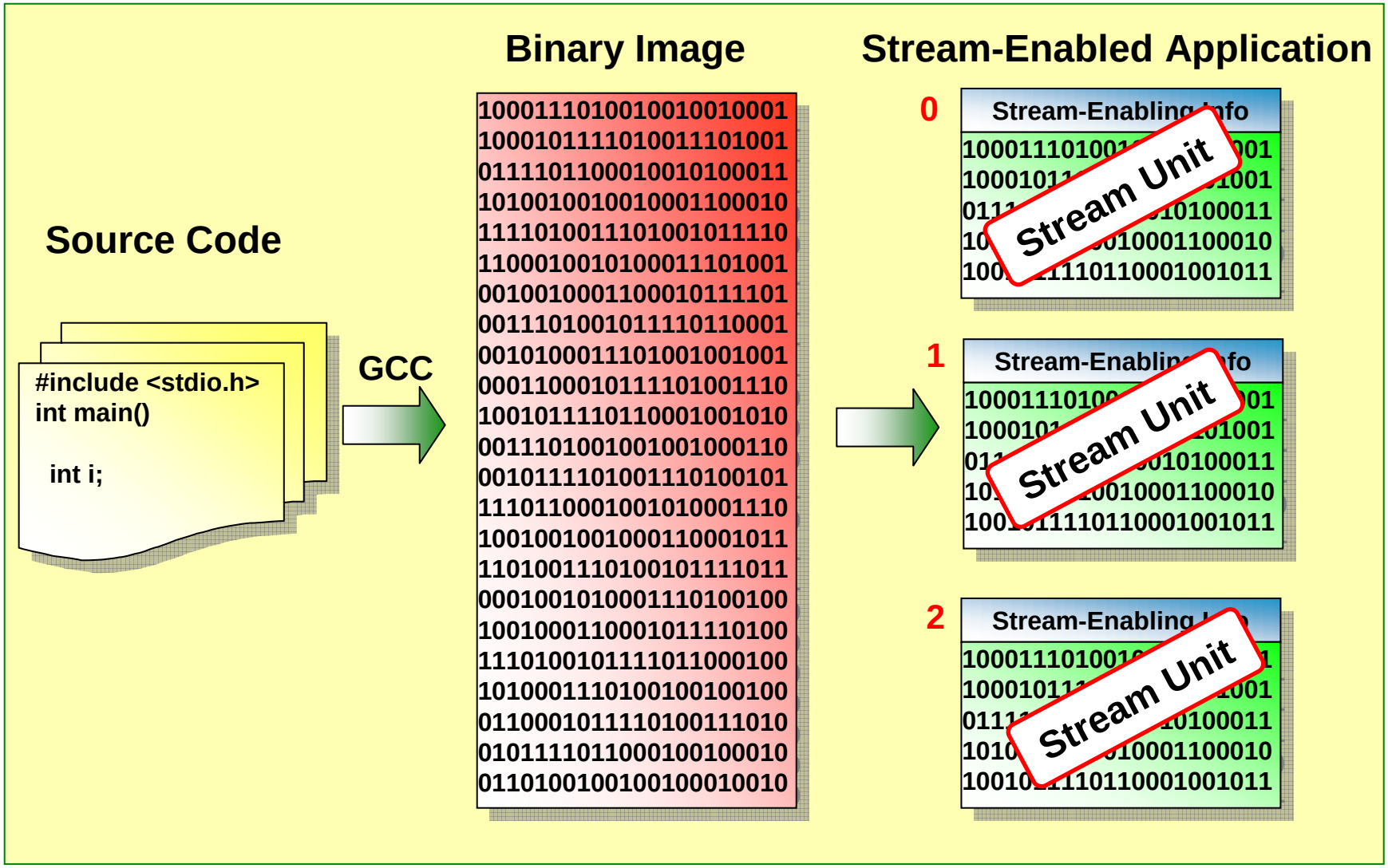
Stream-Enabled Program Files

- **Code generation**
- **Code modification**
- **Code profiling**

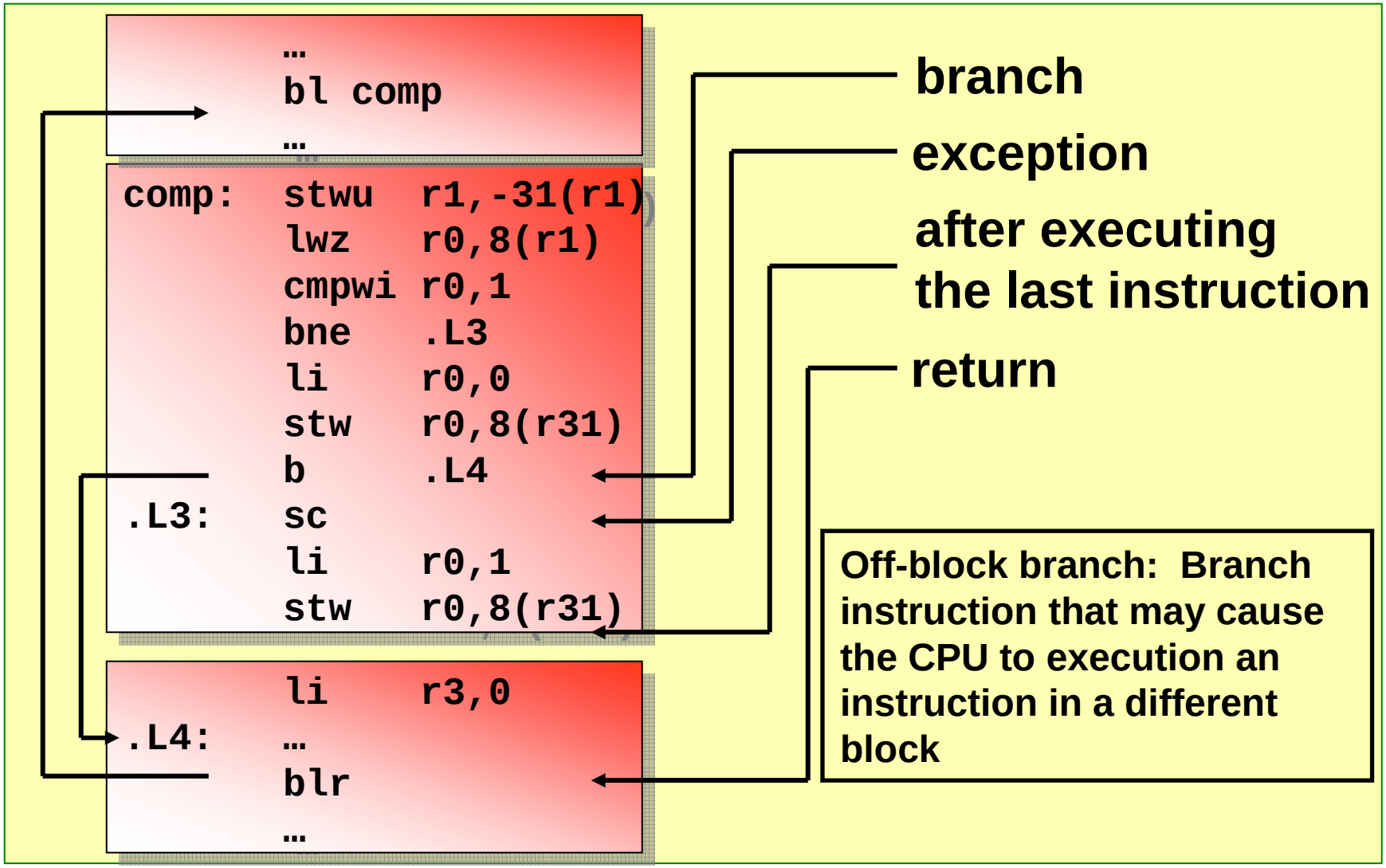
System Overview



Generating Blocks



Exiting and Entering a Block



Code Generation

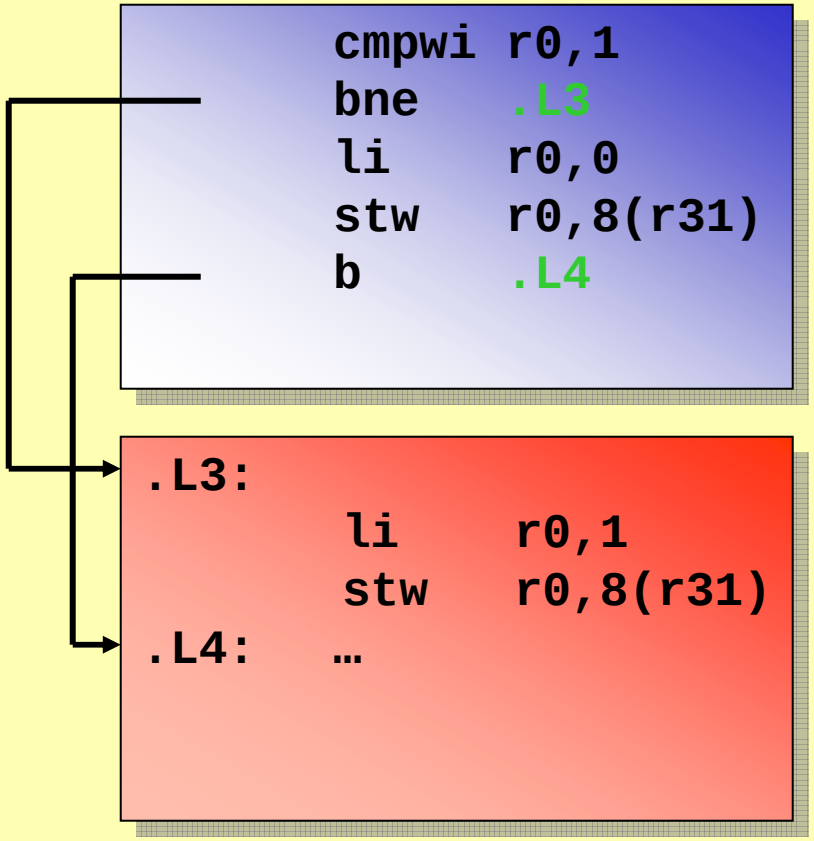
- **Preventing the execution of non-existing code**
  - **Static branches**
  - **Dynamic branches (return, function pointer)**
  - **Exception instructions**
  - **Last instruction of the block**
- **Coping with non-interruptible sections**
  - **Stream execution code prior to the current block**
- **Generating stream-enabling information**
  - **Location of the off-block branches**
  - **Branch number assigned sequentially**

Code Modification

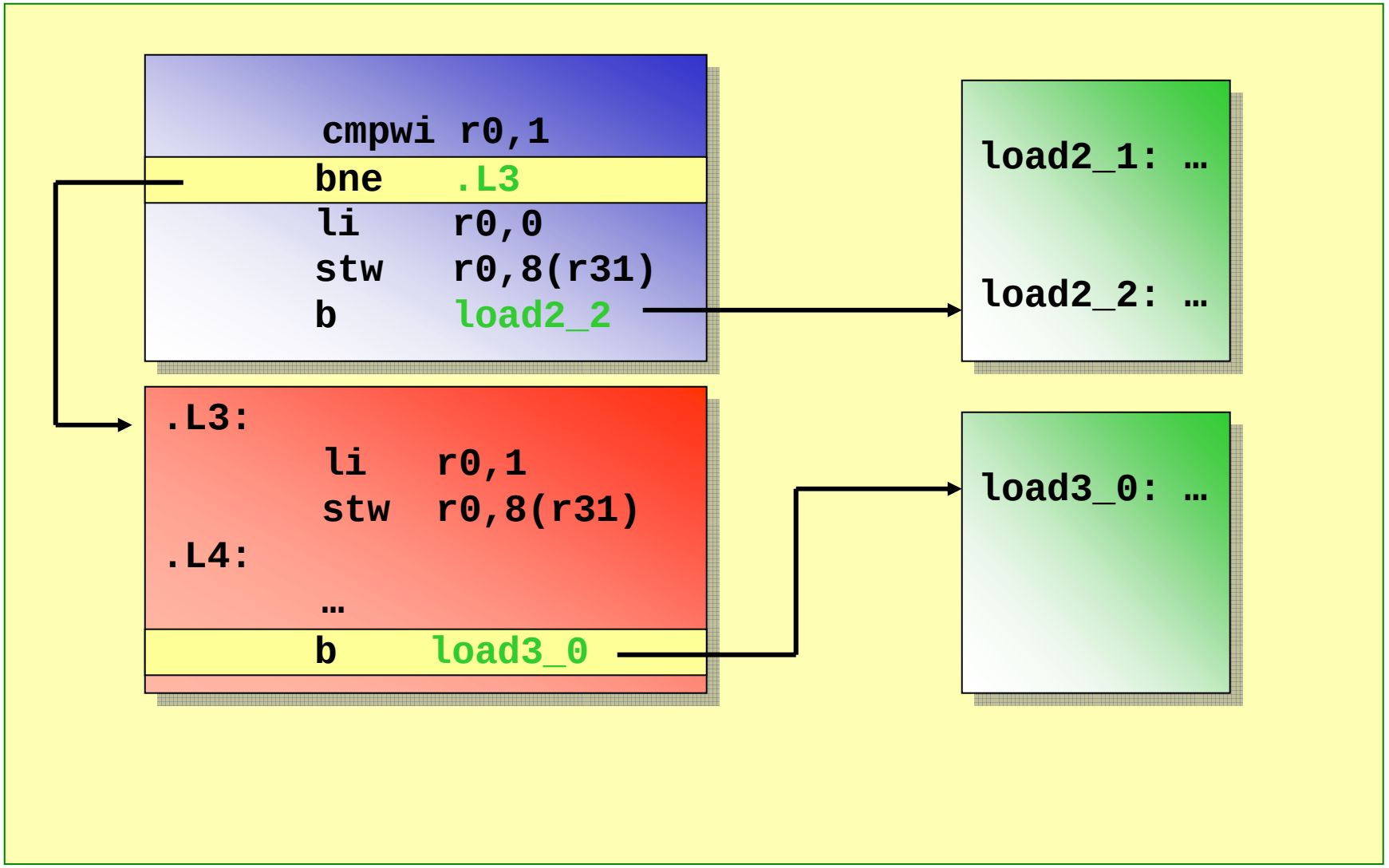
- **Load time code modification**
  - **Modifies off-block branches to jump to the branch table**
  - **Stores in off-block branch information in Branch Info Table**
- **Run time code modification**
  - **Modifies the off-block branches to jump to the intended code**

Code Modification Example (1)

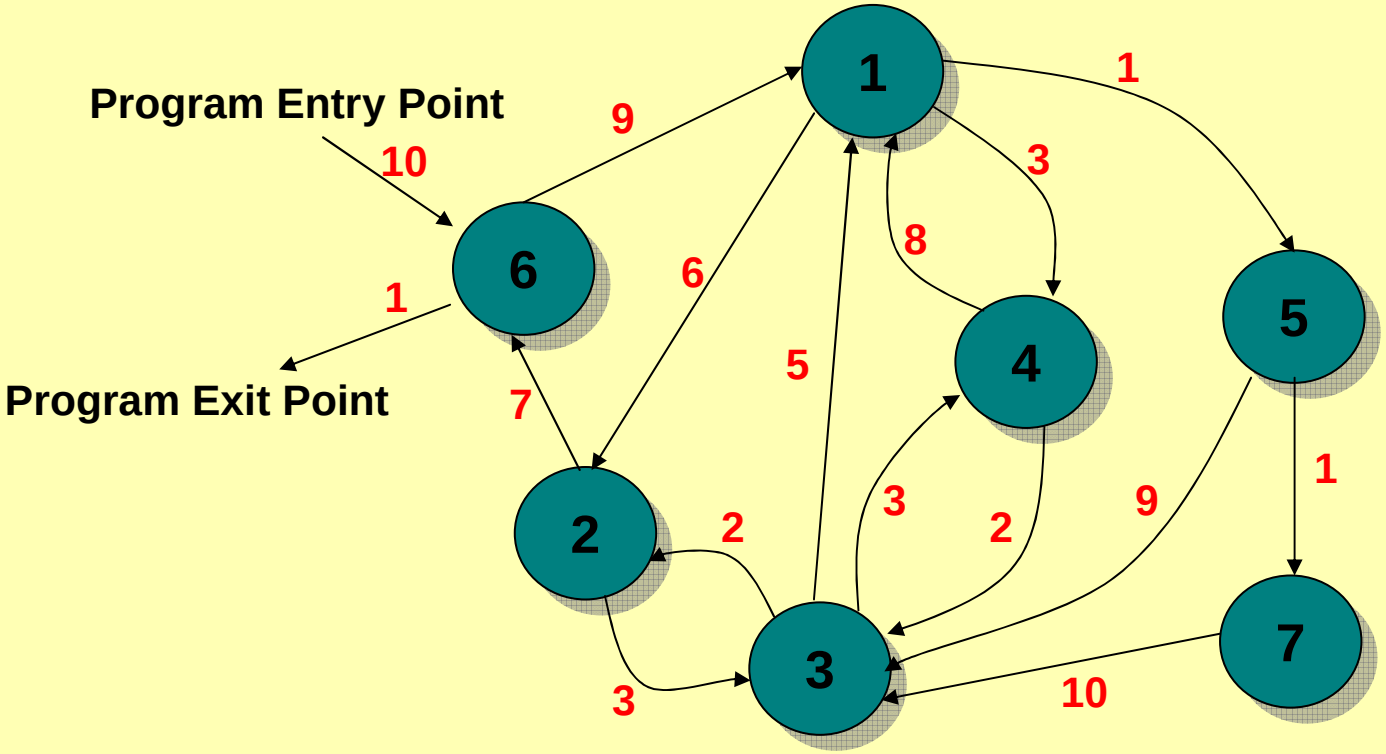
```
if (i==1)
  i=0;
else
  i=1;
```



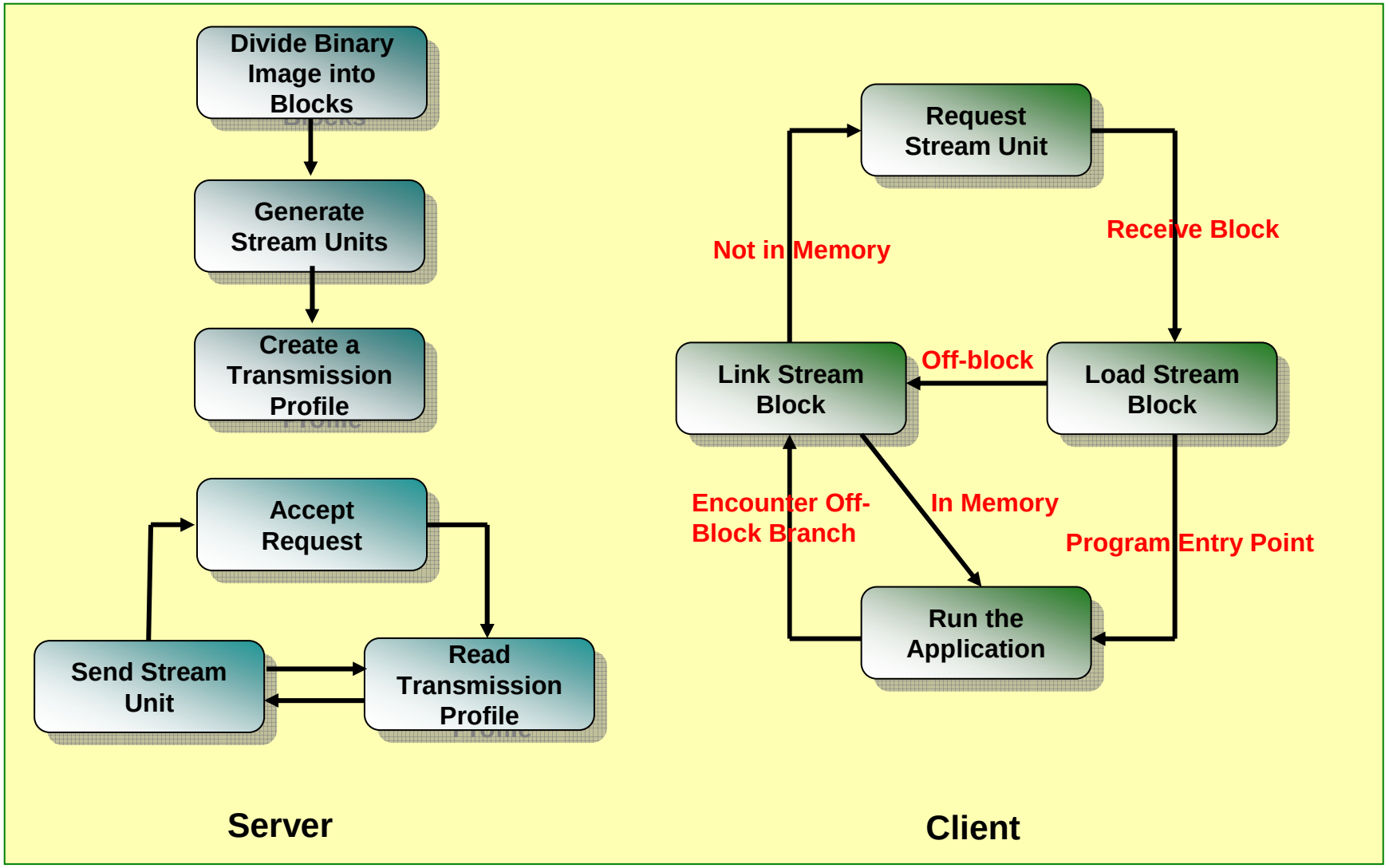
Code Modification Example (2)



Code Profiling



Recap: Block Streaming for Program File

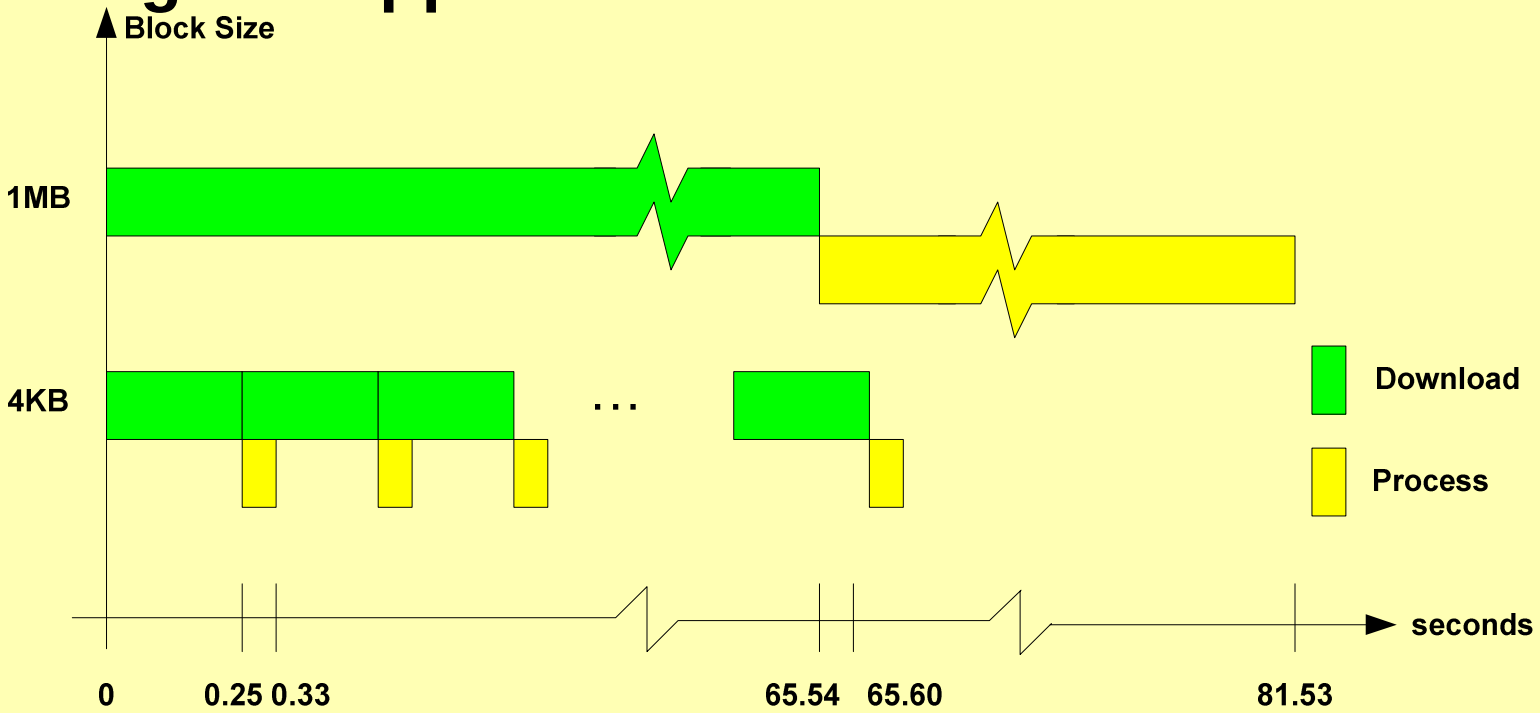


Outline

- Introduction
- Related Work
- Block Streaming
- Stream-Enabled Program Files
- **Stream-enabled File I/O**
- Performance Enhancement
- Experiments and Results
- Conclusion

Stream-Enabled File I/O

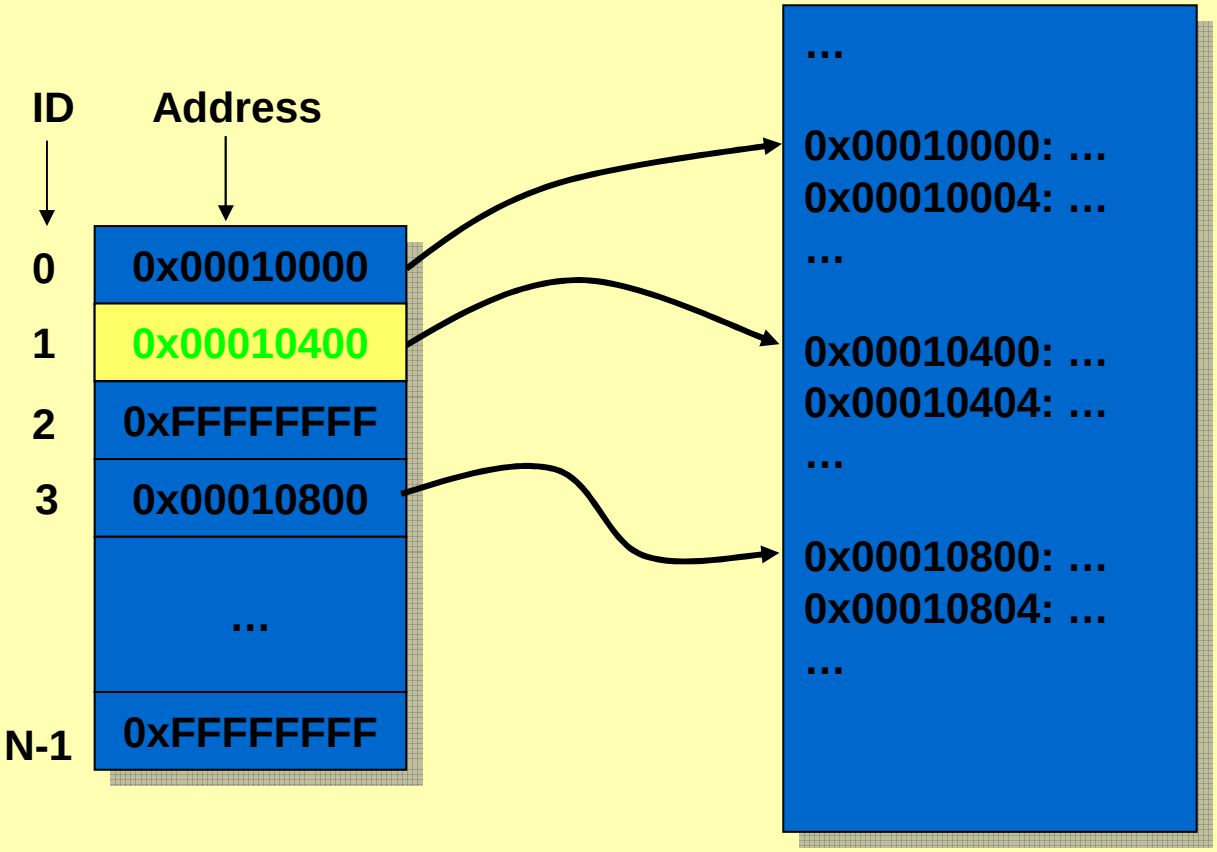
- To reduce file I/O operation latency
- Motivation
  - A game application renders a 1MB scene



Stream Units for Data Block

- Generate stream units by dividing file into fixed size blocks
- Create transmission profile by profiling data
- Provide SIO function calls
  - sio_open()
  - sio_read()
  - sio_write()
  - sio_lseek()
  - sio_close()

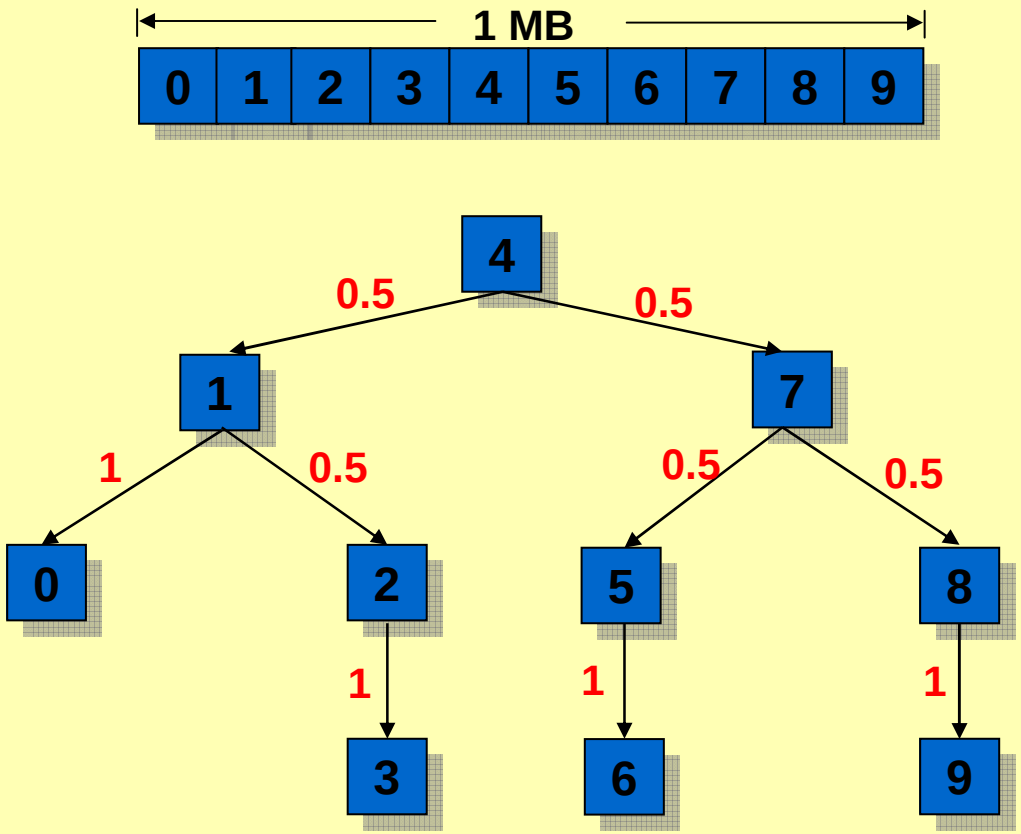
Block Table


ID
Address
0
1
2
3
N-1
0x00010000
0x00010400
0xFFFFFFFF
0x00010800
...
0xFFFFFFFF
...
0x00010000: ...
0x00010004: ...
...
0x00010400: ...
0x00010404: ...
...
0x00010800: ...
0x00010804: ...
...

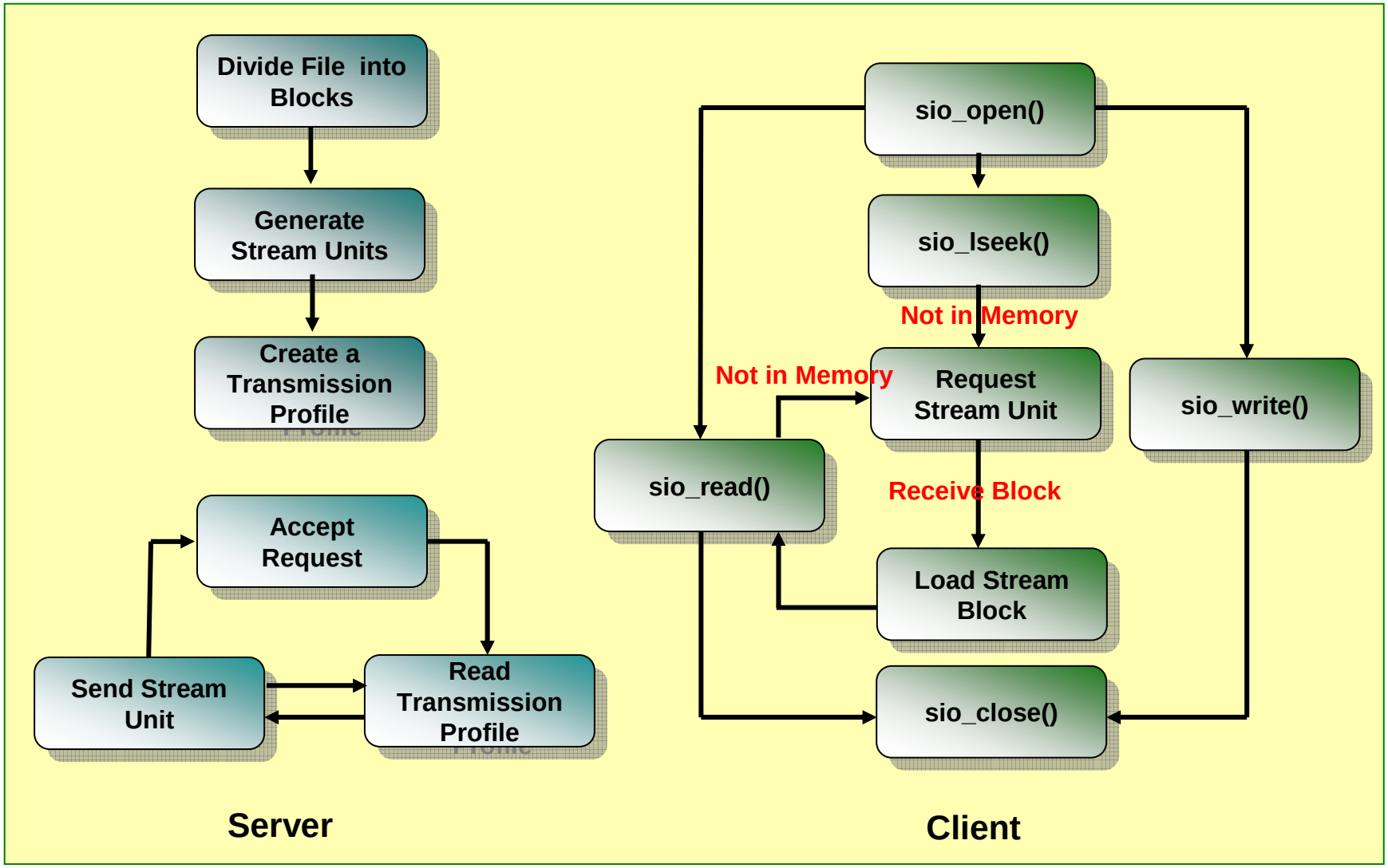
Data Profiling Example (1)

- 1 MB data file, data sorted in ascending order
- Divide the file into 10 equal-size blocks
- A database application searches for a record using a binary search algorithm
- Create a transmission profile according to the binary search algorithm
- Assume that the record is in block 5

Data Profiling Example (2)



Recap: Block Streaming for Data File



Outline

- Introduction
- Related Work
- Block Streaming
- Stream-Enabled Program Files
- Stream-Enabled File I/O
- **Performance Enhancement**
- Experiments and Results
- Conclusion

Performance Enhancement

- **Objectives**
  - **To reduce occurrence of application suspensions**
  - **To support small memory footprint embedded devices**
- **Code transformation**
  - **Enforcing block boundaries**
  - **Remapping functions**
- **Steam unit removal**
  - **Unlinking mechanism**
  - **Stream unit replacement**
- **Requirement**
  - **Fixed sized stream blocks**

Enforcing Block Boundaries Example

```
fn1:    stwu 1,-31(1)
        stw 3,8(1)
        lwz 0,8(1)
        …
        blr
fn2:    stwu 1,-31(1)
        stw  3,8(1)
        li 0,1
        …
        blr
fn3:    …
```

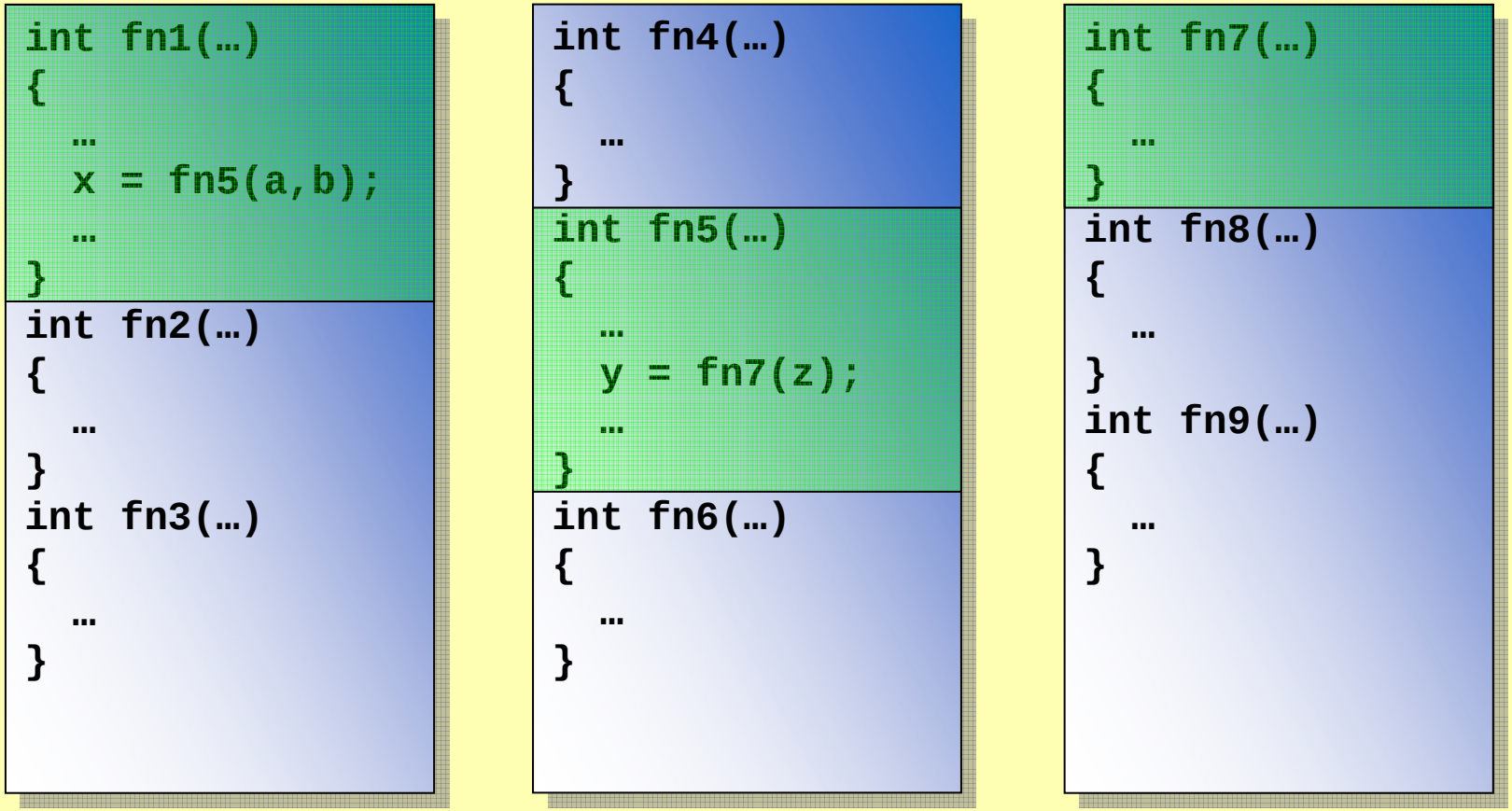
```
fn1:    stwu 1,-31(1)
        stw 3,8(1)
        lwz 0,8(1)
        …
        blr

fn2:    stwu 1,-31(1)
        stw  3,8(1)
        li 0,1
        …
        blr
fn3:    …
```

Remapping Functions

- **Observations**
  - **Programmer places functions in the file randomly**
  - **Compiler places the functions the same order as written**
  - **Program jumps from block to block**
  - **Higher occurrence of software suspensions**

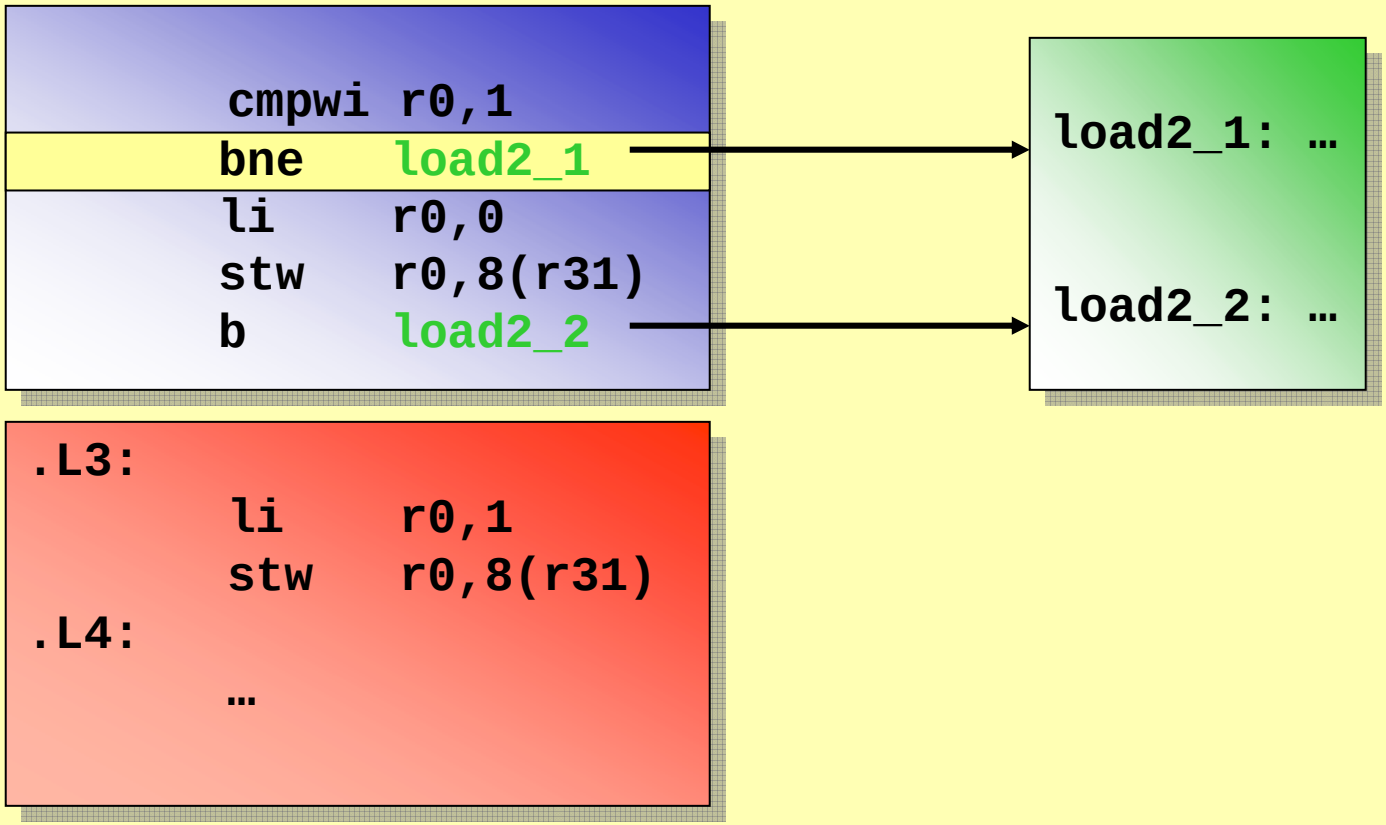
Remapping Function Example



Unlinking Mechanism

- **Linking**
  - **Run efficiently, no code checking**
- **Unlinking**
  - **Remove blocks**
  - **Need to know location of incoming branches to the block to be removed**

Unlinking Mechanism Example

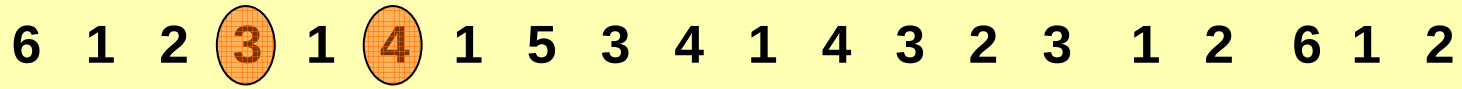


Stream Unit Replacement

- **Objective**
  - **Reduce number of retransmissions**
- **Example**
  - **Game application (e.g., Doom)**
    - **6 MB**
    - **6 blocks, 1MB each**
  - **PDA with 3 MB memory available**
  - **3 Blocks, 1 MB each**

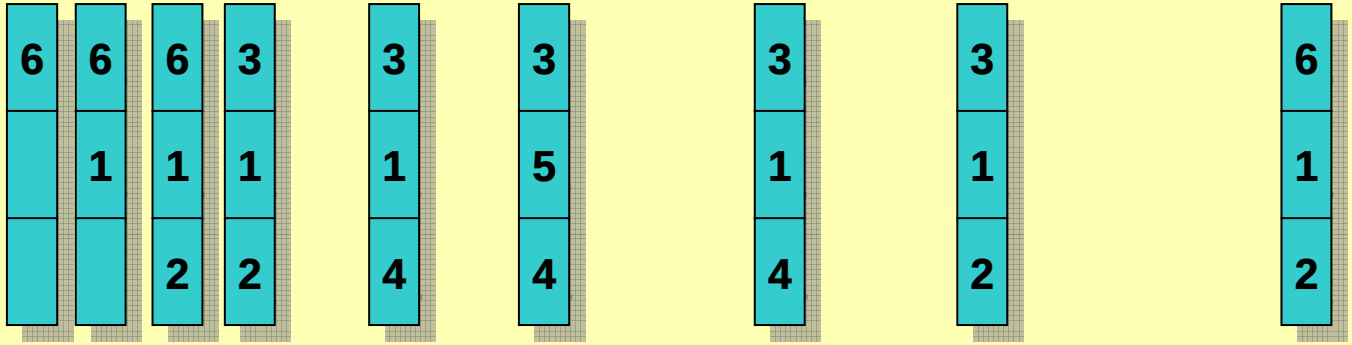
Stream Unit Replacement Example

Execution profile:



Transmission profile:

Client memory:



9 occurrences of application suspension for demand loading
potentially 6 occurrences with block streaming

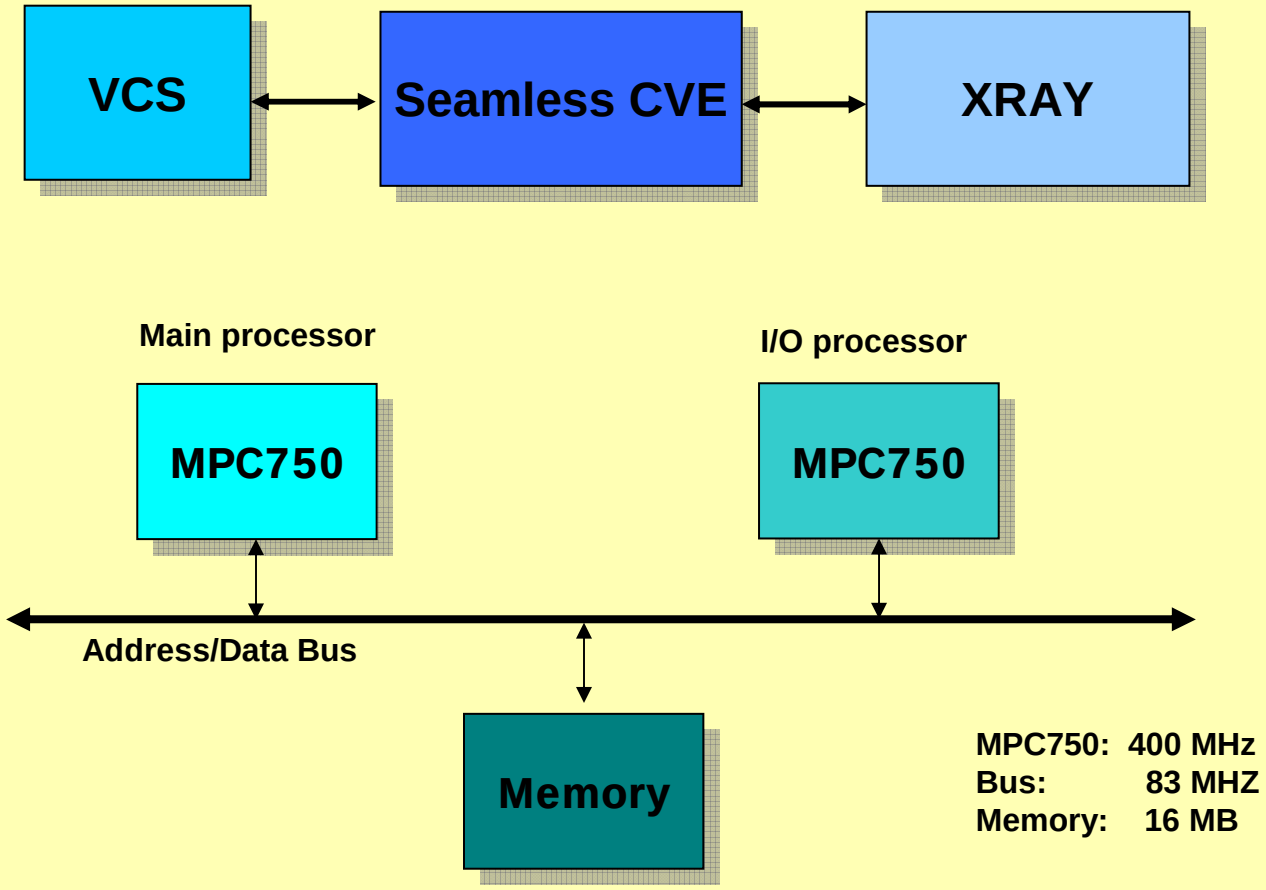
Outline

- Introduction
- Related Work
- Block Streaming
- Stream-Enabled Program Files
- Stream-Enabled File I/O
- Performance Enhancement
- **Experiments and Results**
- Conclusion

Experiments and Results

- **Hardware setup**
- **Stream-enabled program file (SPF)**
- **Stream-enabled file I/O (SIO)**
- **Stream-enable program with stream-enabled file I/O**

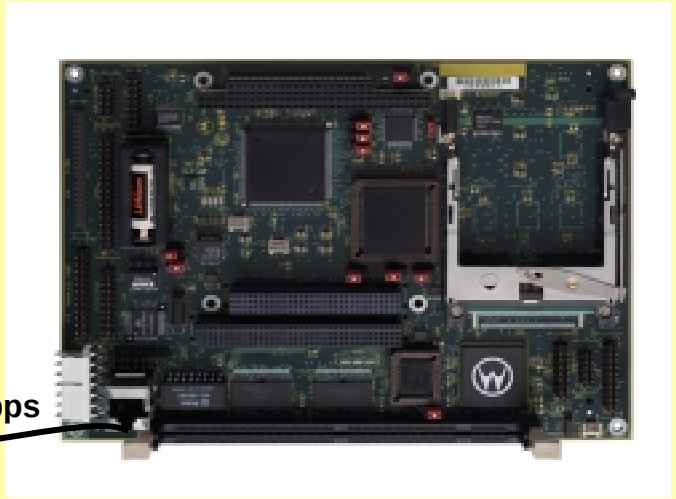
Simulation Environment



MBX860 Board Environment

Network Cloud

10Mbps

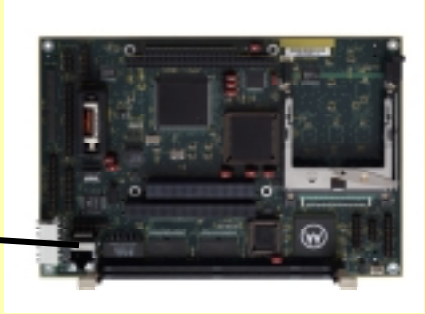


PC:
Linux
Traffic Shaper

MBX860:
- **PowerPC 860, 40MHz**
- **10BaseT Ethernet**
- **4 MB DRAM, 2 MB Flash**
- **Linux 2.4.21**

Code Size

Implementation	C lines
softstream server	≈ 3400
softstream client	≈ 1400
softstream loader/linker	≈ 1300
stream-enabled file I/O	≈ 1500
softstream generator	≈ 2200



Server:
softstream server
softstream generator

Client:
softstream client
softstream loader/linker
stream-enabled file I/O

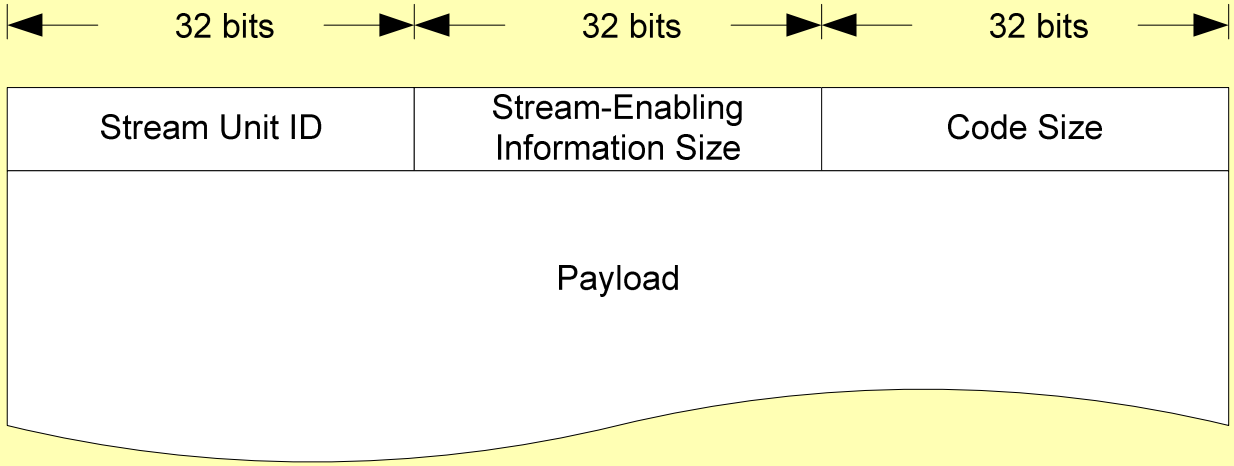
SPF Overhead (1)

- **Overhead per off-block branch**
  - **Bandwidth 4 bytes (location of the branch)**
  - **Memory 20 bytes (12 bytes for block table, 4 bytes for instruction, 4 bytes for location)**

Type of overhead	Overhead per off-block branch
Bandwidth	4 bytes
Memory	20 bytes

SPF Overhead (2)

- **Overhead per block**
  - **Bandwidth: 12 bytes for headers + 4*n**
  - **Memory: 20*n**
  - **n = number of off-block branches**



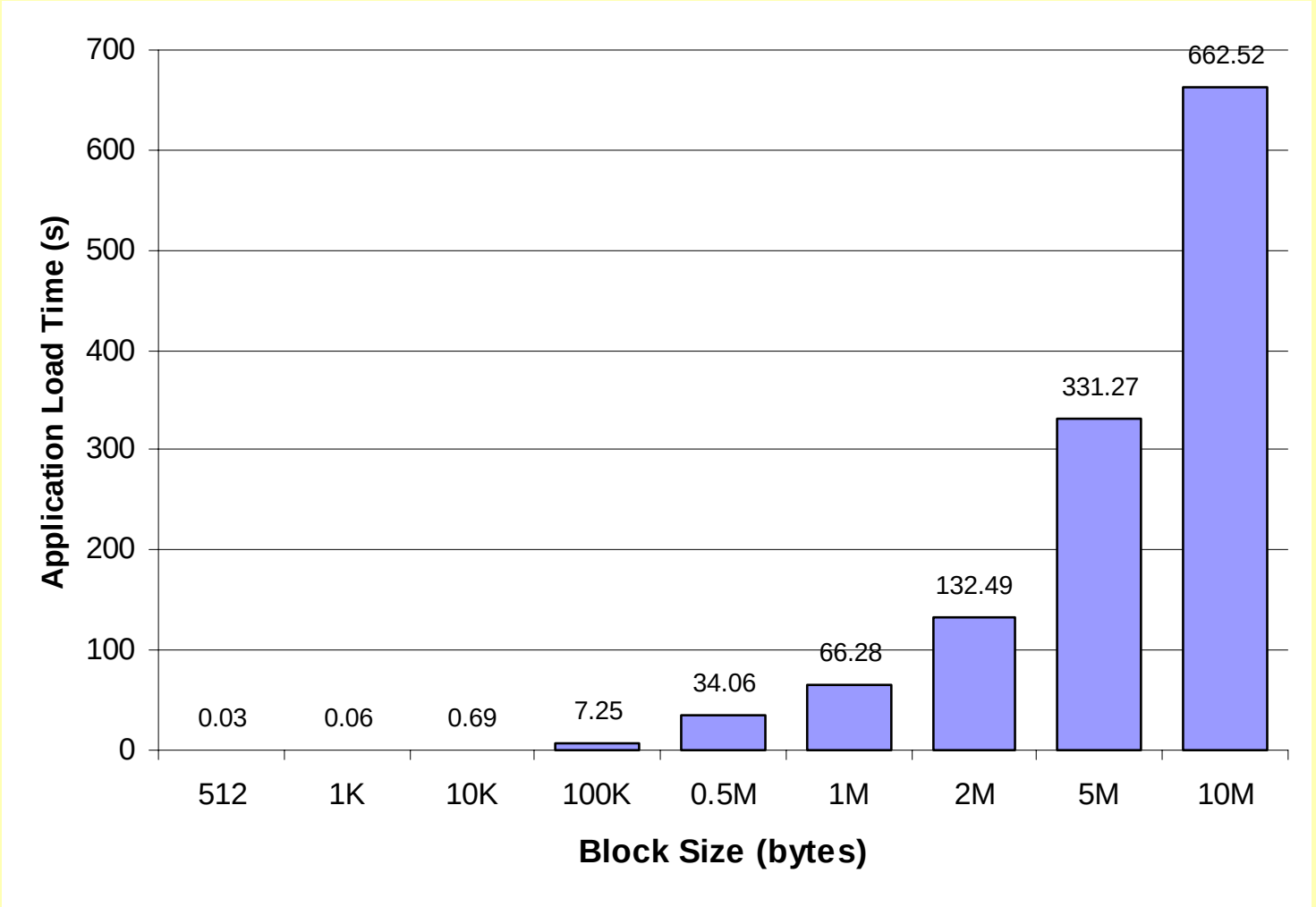
SPF Simulation Scenario

- Adaptive autonomous robot exploration
- Impossible to write and load software for all possible environments
- The mission control needs to update the robot software over a 128Kbps link
- The new code is 10MB
- The robot does not need all 10MB initially
- The robot must run the software to react to the new environment within 120 s

SPF Simulation Results

Block size (bytes)	Total # of blocks	Added code/block	Load time (s)
10M	1	0.0003%	655.36
5M	2	0.0007%	327.68
2M	5	0.0017%	131.07
1M	10	0.0034%	65.54
0.5M	20	0.0069%	32.77
100K	103	0.0352%	6.40
10K	1024	0.3516%	0.64
1K	10240	3.5156%	0.06
512	20480	7.0313%	0.03

SPF MBX860 Board Results

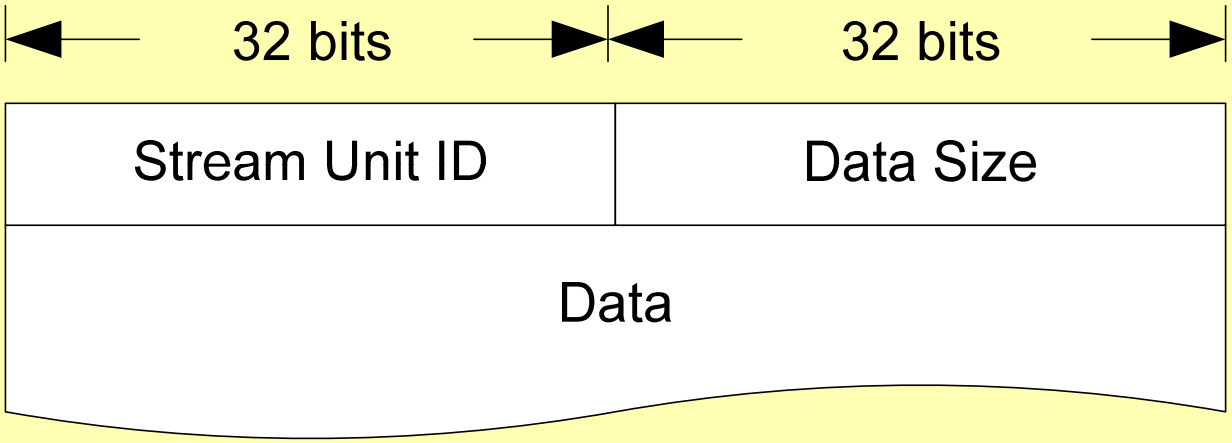


Stream-Enabled Program File Results

- Sending the whole software takes over 10 minutes: the deadline is missed
- Using software streaming with the first blocks of size of 1MB, the new software can be executed within 66 seconds: the deadline is met
- The application load time improves by a factor of ≈ 10X
- Function streaming can potentially achieve the same result
  - But function streaming lacks file I/O support

Stream-Enabled File I/O (SIO)

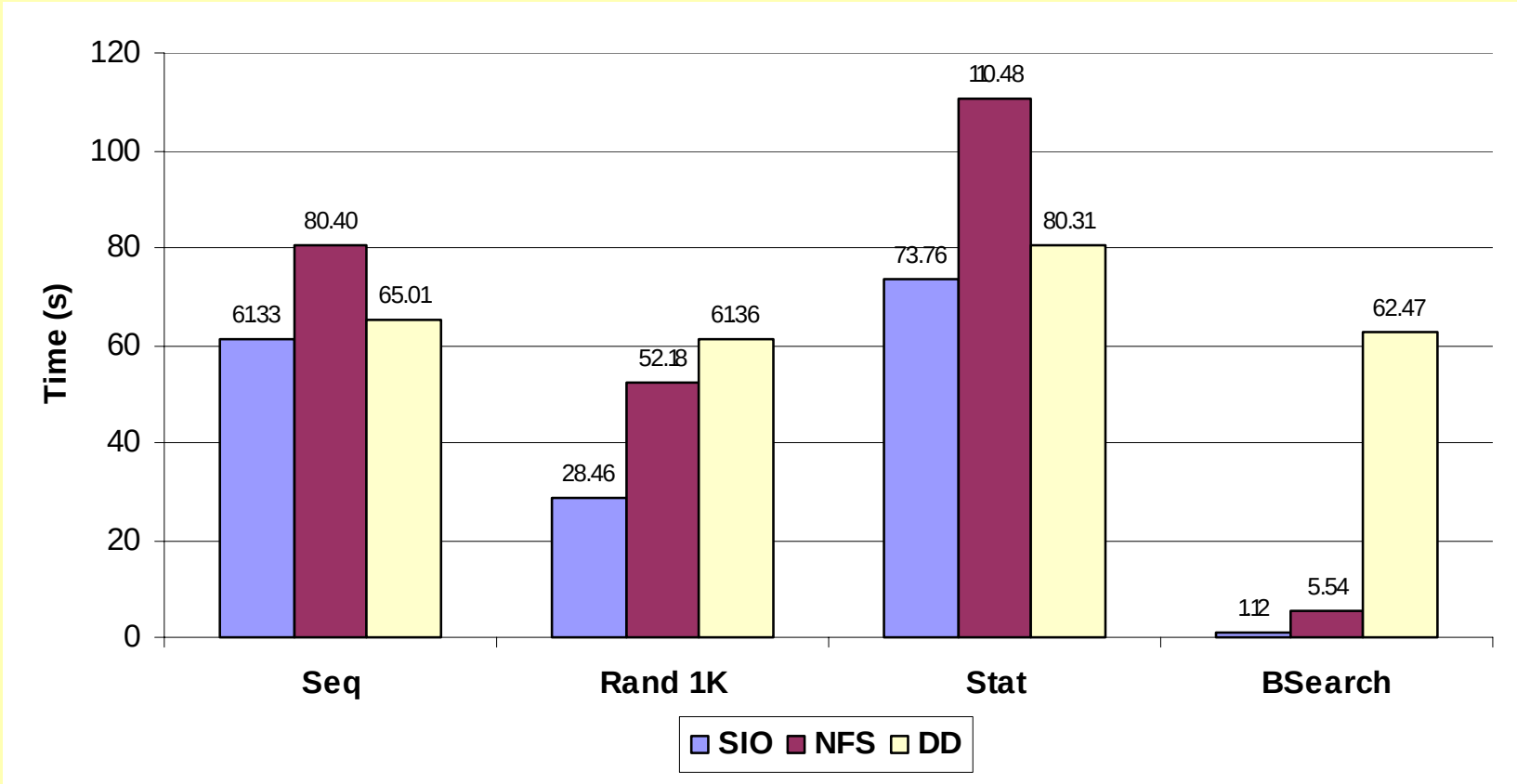
- **Overhead per block**
  - **Bandwidth 8 bytes (Stream Unit ID, Data Size)**
  - **Memory 4 bytes (Entry in Block Table)**



SIO Board Experiment (1)

- File size 1 MB
- Benchmarks
  - Seq: read data sequentially
  - Rand 1K: read 1KB randomly
  - Stat: calculate various statistical values of distinct pieces the data
  - BSearch: find a specific value in the file using a binary search algorithm
- Implementations
  - DD using Linux TCP 1.0 for NET4.0, NFS version 3, SIO

SIO Board Results (1)



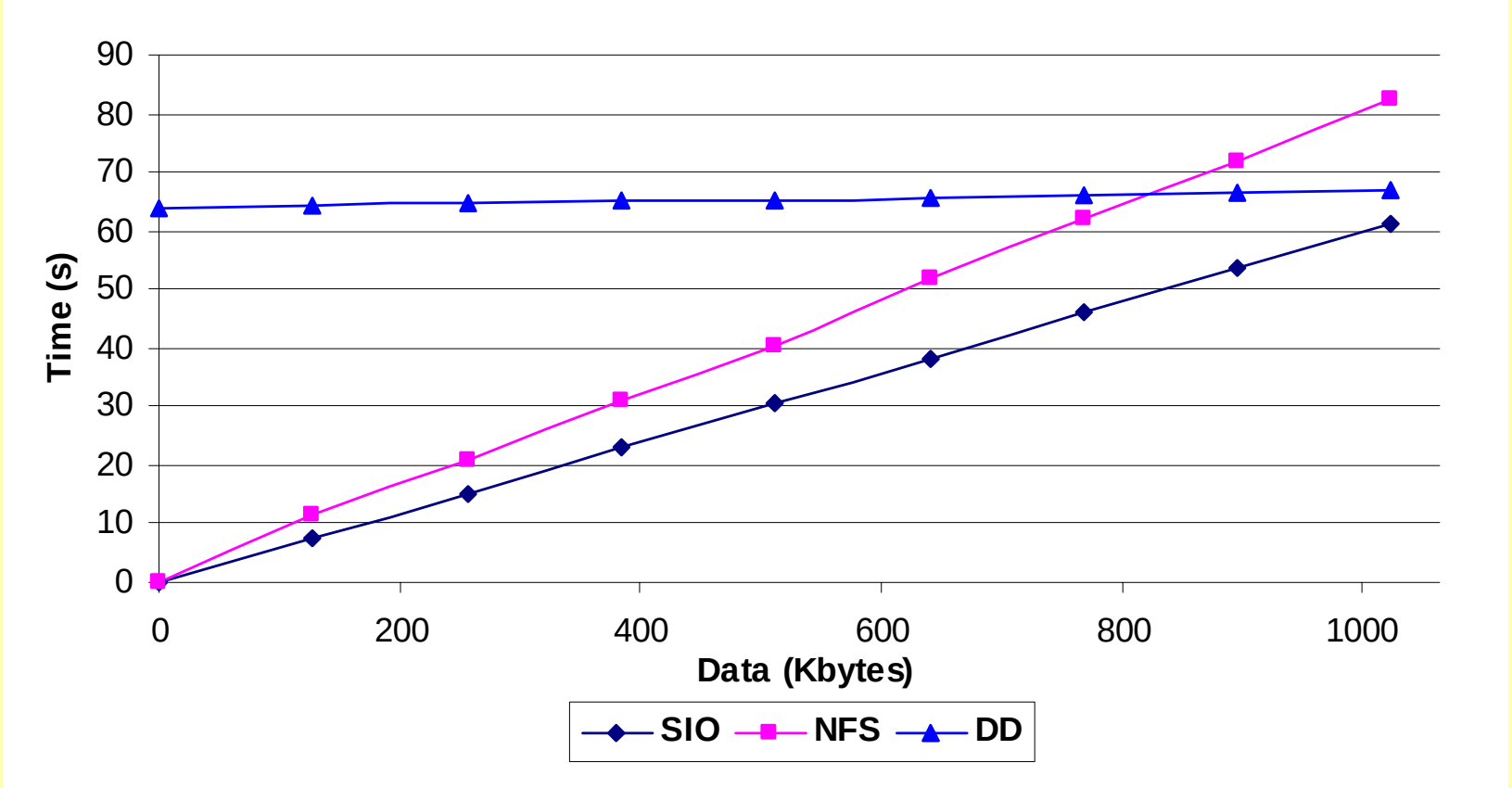
Up to 55X faster

SIO Board Experiment (2)

- **Data acquisition**
  - **Read a certain amount of data from a 1 MB file**
  - **Link speed 128 Kbps**

SIO Board Results (2)

Time to acquire a certain amount of data

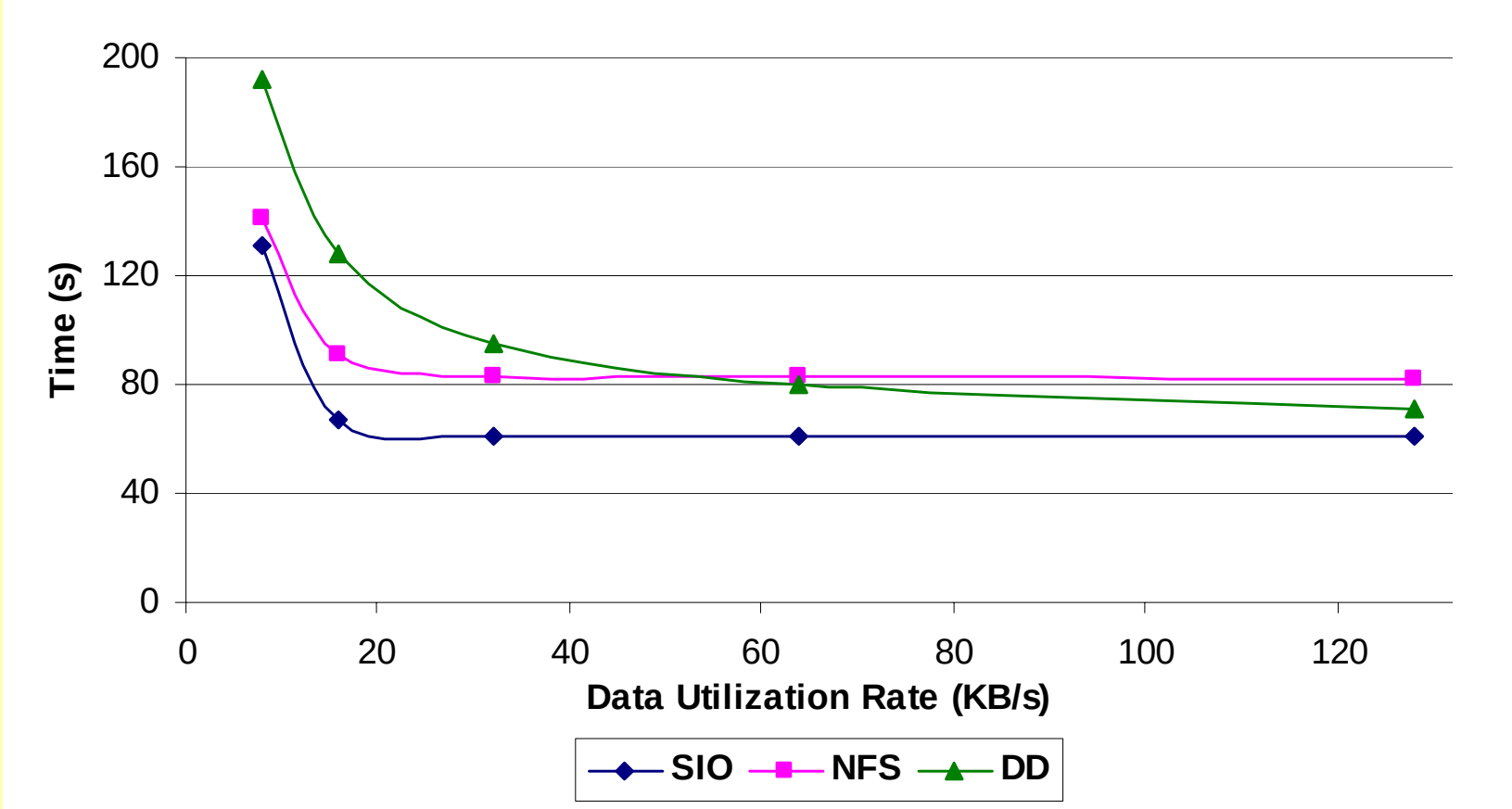


SIO Board Experiment (3)

- **Data utilization (Kbytes per second)**
  - **How fast data being consumed**
  - **Process a 1 MB file using various data utilization rates**
  - **Link speed 128 Kbps**

SIO Board Results (3)

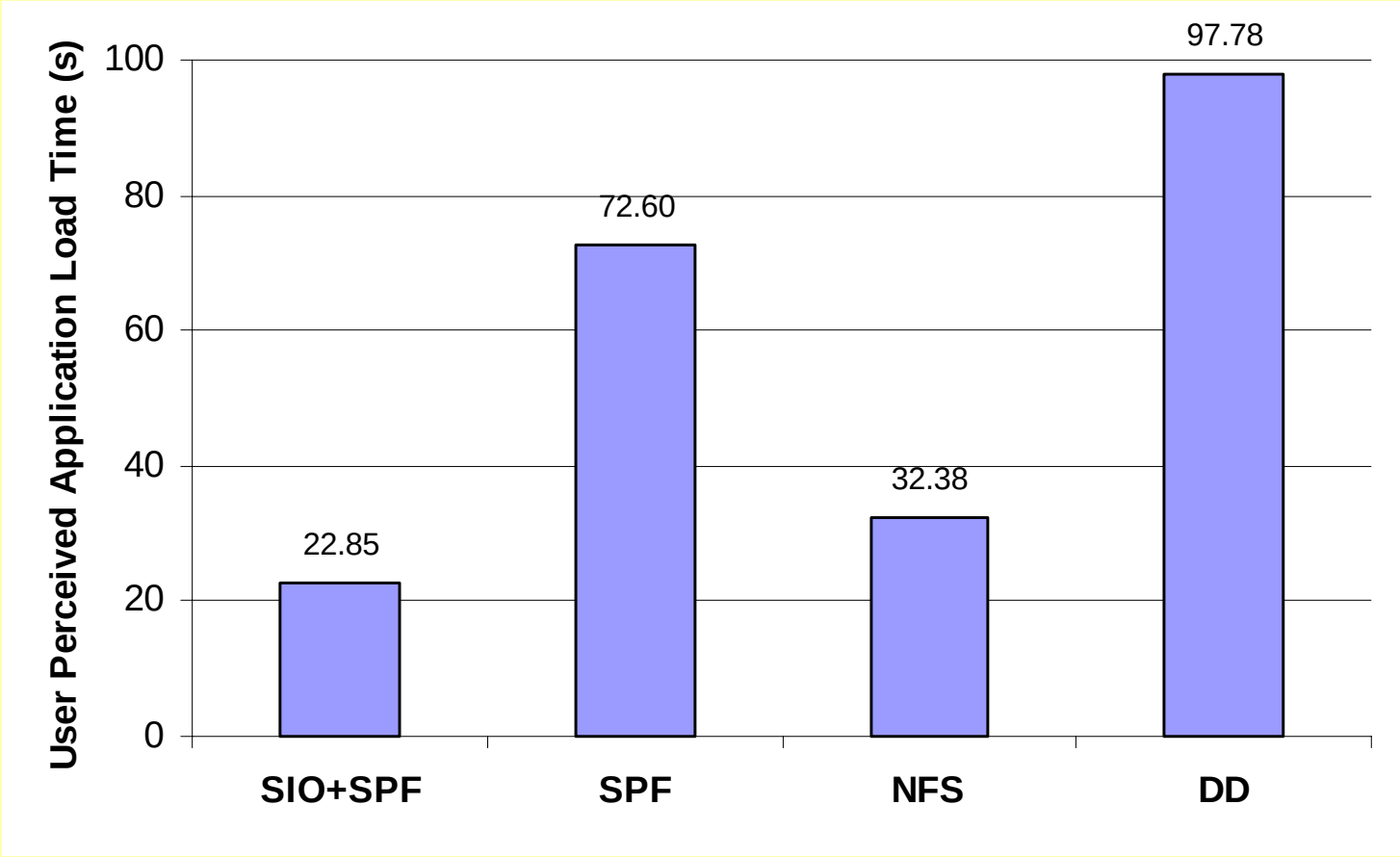
The amount of time it takes to process a 1 MB file



Stream-Enabled Application Experiment

- Game application
- Program size 512 KB
- Data size 1 MB
- 128 KB code and 256 KB data needed for the first scene
- Stream-enabled program file (SPF) embeds data inside program
- Implementation: SIO+SPF, SPF, NFS, DD

Stream-Enabled Application Results



Conclusion

- Reduce application load time by enabling execution while transferring (10X)
- Lower application suspension time by profiling
- Reduce the occurrence of application suspensions
- Support small memory footprint embedded devices
- Provide stream-enabled file I/O (55X)

Publications

- **Kuacharoen, P. and Mooney, V., "Memory management for embedded devices using software streaming," to be published in *Proceedings of the Mobility Conference & Exhibition*, Aug. 2004.**
- **Akgul, B., Mooney, V., Thane, H., and Kuacharoen, P., "Hardware Support for Priority Inheritance," in *Proceedings of the IEEE Real-Time Systems Symposium*, pp.246-254, Dec. 2003.**
- **Kuacharoen, P., Mooney, V., and Madisetti, V., "Software streaming via block streaming," in the book *Embedded Software for SoC*, edited by Jerraya, A., Yoo, S., Verkest, D. and Wehn, N., Boston, MA: Kluwer Academic Publishers, pp. 435-448, Sep. 2003.**
- **Kuacharoen, P., Mooney, V., and Madisetti, V., "Software streaming via block streaming," in *Proceedings of the Design Automation and Test in Europe*, pp. 912-917, Mar. 2003.**
- **Kuacharoen, P., Shalan, M., and Mooney, V., "A congurable hardware scheduler for real-time systems," in *Proceedings of the International Conference on Engineering of Recongurable Systems and Algorithms*, pp. 96-101, June 2003.**
- **Kuacharoen, P., Akgul, T., Mooney, V., and Madisetti, V., "Adaptability, extensibility, and exibility in real-time operating systems," in *Proceedings of the EUROMICRO Symposium on Digital Systems Design*, pp. 400-405, Sep. 2001.**
- **Akgul, T., Kuacharoen, P., Mooney, V., and Madisetti, V., "A debugger RTOS for embedded systems," in Proceedings of the 27th EUROMICRO Conference, pp. 264-269, Sep. 2001.**

Patents

- **Kuacharoen, P., Mooney, V., and Madisetti, V., "Methods and systems for transmitting application software," *U.S. Patent Application 20040006637*, Jan. 2004.**
- **Kuacharoen, P., Akgul, T., Mooney, V., and Madisetti, V., "Dynamic operating system," *U.S. Patent Application 20030074487*, Apr. 2003.**
- **Akgul, T., Kuacharoen, P., Mooney, V., and Madisetti, V., "Debugger operating system for embedded systems," *U.S. Patent Application 20030074650*, Apr. 2003.**

Security: Issues not Addressed

- **Network security**
  - **Stream applications from trusted site**
  - **SSL**
  - **Certificate**
- **Memory protection**

Questions?