CS302

Microservices & RPC

Spring 2025

Arkaprava Basu & Babak Falsafi

parsa.epfl.ch/course-info/cs302

Adapted from slides originally developed by Profs. Falsafi, Fatahalian, Mowry, Wenisch of CMU, Michigan

Where are We?

M	T	W	T	F
17-Feb	18-Feb	19-Feb	20-Feb	21-Feb
24-Feb	25-Feb	26-Feb	27-Feb	28-Feb
3-Mar	4-Mar	5-Mar	6-Mar	7-Mar
10-Mar	11-Mar	12-Mar	13-Mar	14-Mar
17-Mar	18-Mar	19-Mar	20-Mar	21-Mar
24-Mar	25-Mar	26-Mar	27-Mar	28-Mar
31-Mar	1-Apr	2-Apr	3-Apr	4-Apr
7-Apr	8-Apr	9-Apr	10-Apr	11-Apr
14-Apr	15-Apr		17-Apr	18-Apr
21-Apr	22-Apr	23-Apr	24-Apr	25-Apr
28-Apr	29-Apr	30-Apr	1-May	2-May
5-May	6-May	7-May	8-May	9-May
12-May	13-May	14-May	15-May	16-May
19-May	20-May	21-May	22-May	23-May
26-May	27-May	28-May	29-May	30-May

- Microservices and RPC
  - Microservices vs Monoliths
  - Communication using RPC
- Exercise session
  - Coroutine examples
- Next Tuesday:
  - Easter break!
- Next lecture:
  - Example RPC (gRPC)

Recap: Web Server Functions (Coroutines lecture)

```
def save_file(req):
    ...

def get_line():
    ...

def sha(req):
    ...

def handle_req(req):
    ...
```

→ Saves text to a file

Recap: Web Server Functions

```
def save_file(req):
    ...

def get_line():
    ...

def sha(req):
    ...

def handle_req(req):
    ...
```

Streams a file line by line

Recap: Web Server Functions

```
def save_file(req):
    ...

def get_line():
    ...

def sha(req):
    ...

def handle_req(req):
    ...
```

Calculates and saves a hash

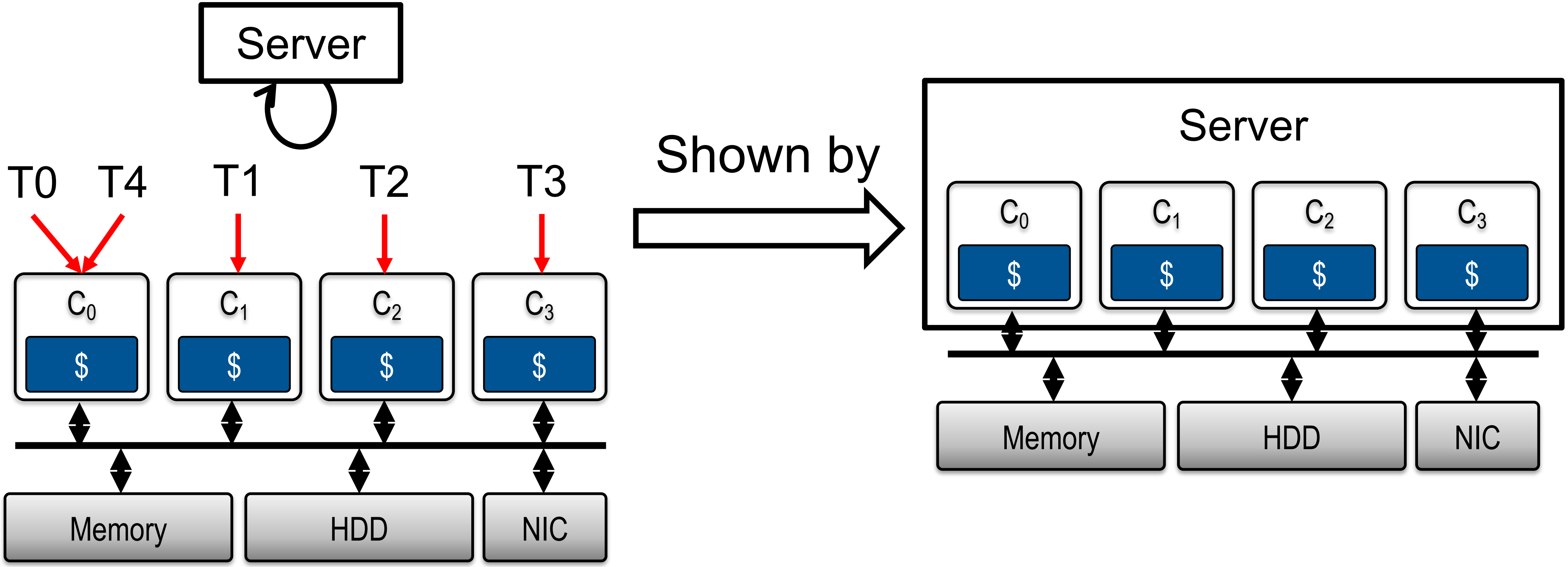
Recap: Web Server Functions

```
def save_file(req):
    ...

def get_line():
    ...

def sha(req):
    ...

def handle_req(req):
    ...
```

Depending on the request type, calls the corresponding function (**`save_file/get_line/sha`**) to execute

Recap: Web Server

- Web server spawns a new thread to handle each request
  - Threads run both in parallel and concurrently (if threads > cores)



Traditional Approach To Writing Server Software

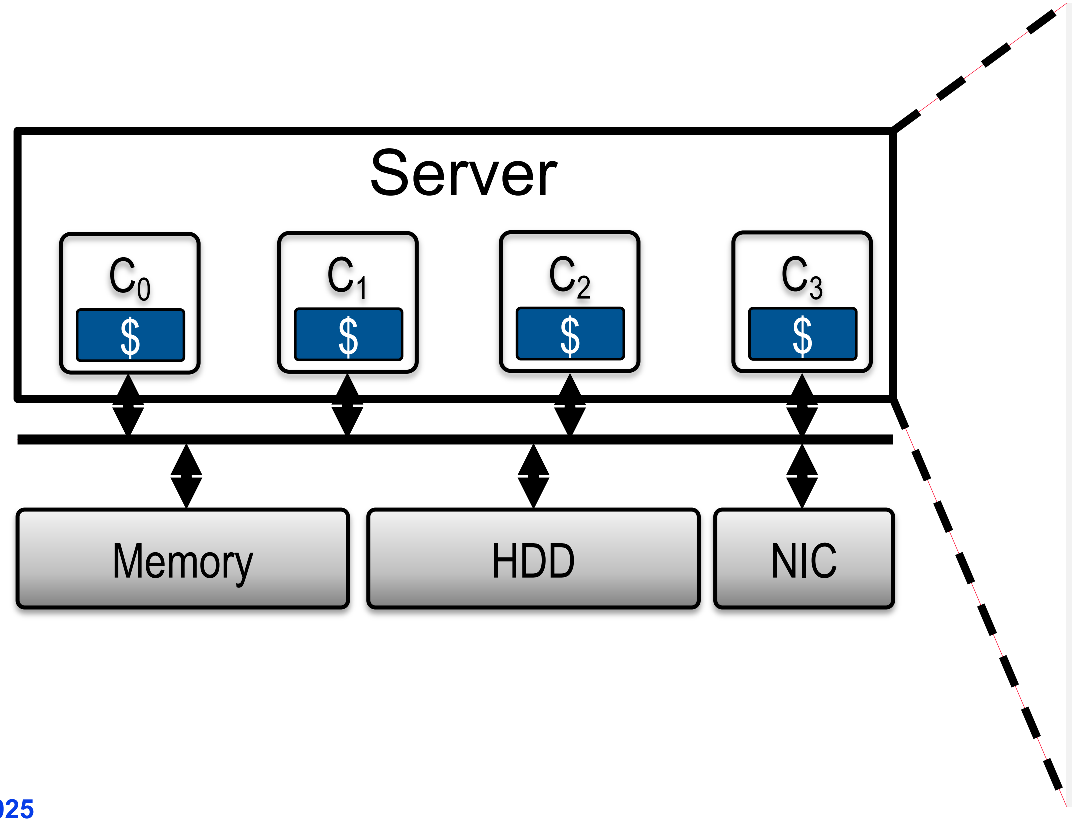
```
def save_file(req):
    ...

def get_line():
    ...

def sha(req):
    ...

def handle_req(req):
    ...
```

Notice all functions are in a single program!

This is the traditional way of writing software we have looked at so far

Monoliths: Traditional Server Software

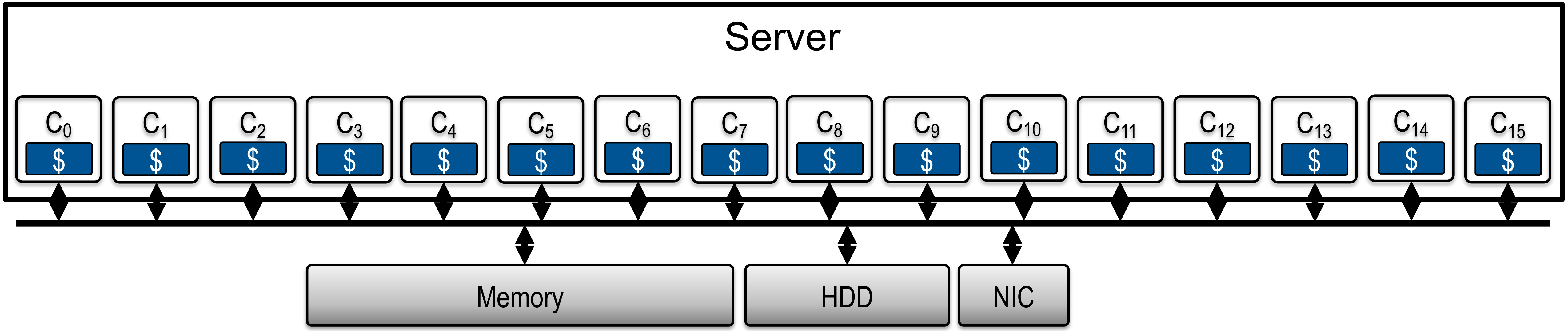
- A monolith is a single application that contains all the functionality
  - All functions are part of the same program and run in the same address space
  - E.g., The web server serving user requests and running on four cores




```
def save_file(req):
    ...

def get_line():
    ...

def sha(req):
    ...

def handle_req(req):
    ...
```

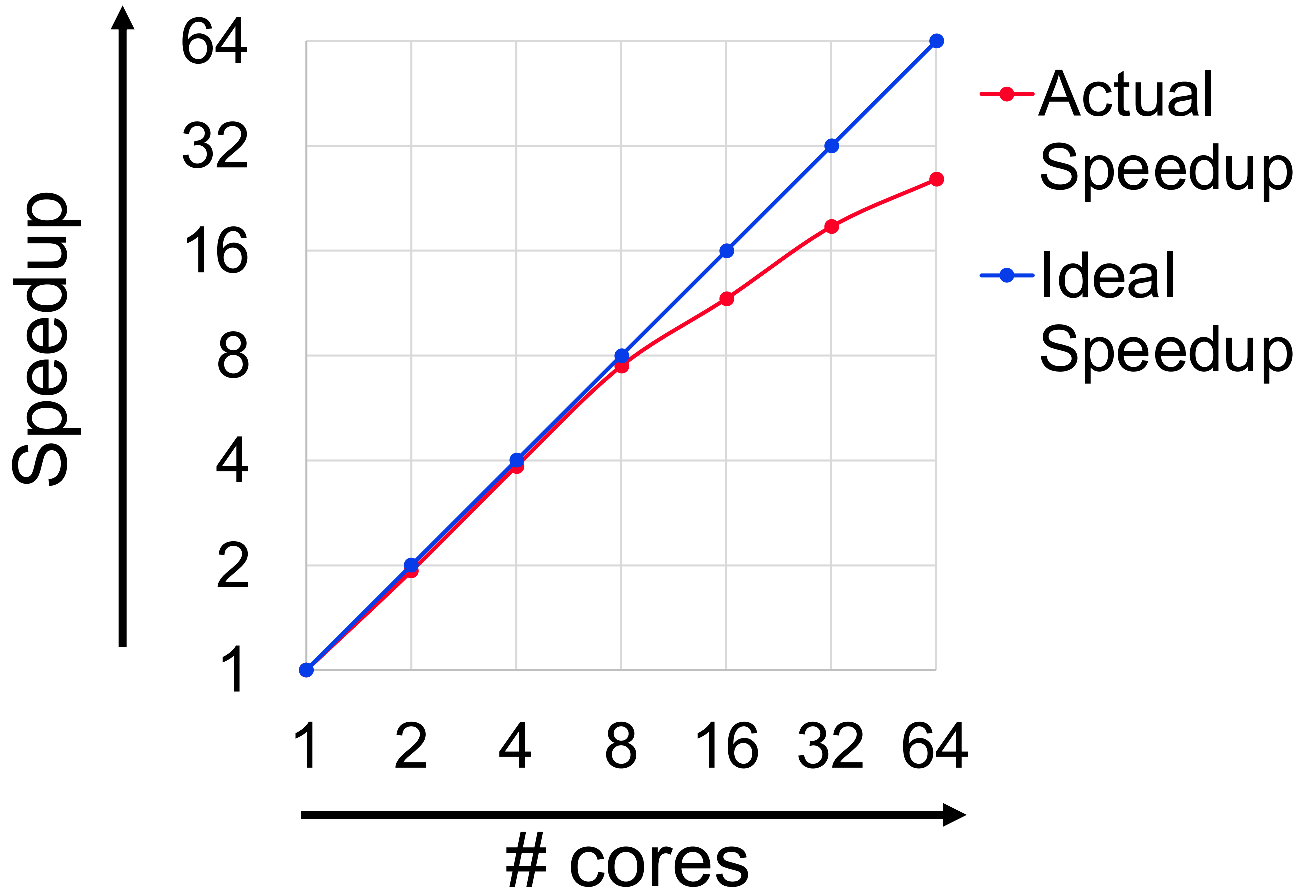
Monoliths Do Not Scale Well

- Real world monoliths are complex and contain several functions
  - Suffer from synchronization overheads among threads
- Overheads limit monoliths from fully utilizing all the cores
  - Synchronization limits performance beyond a few cores

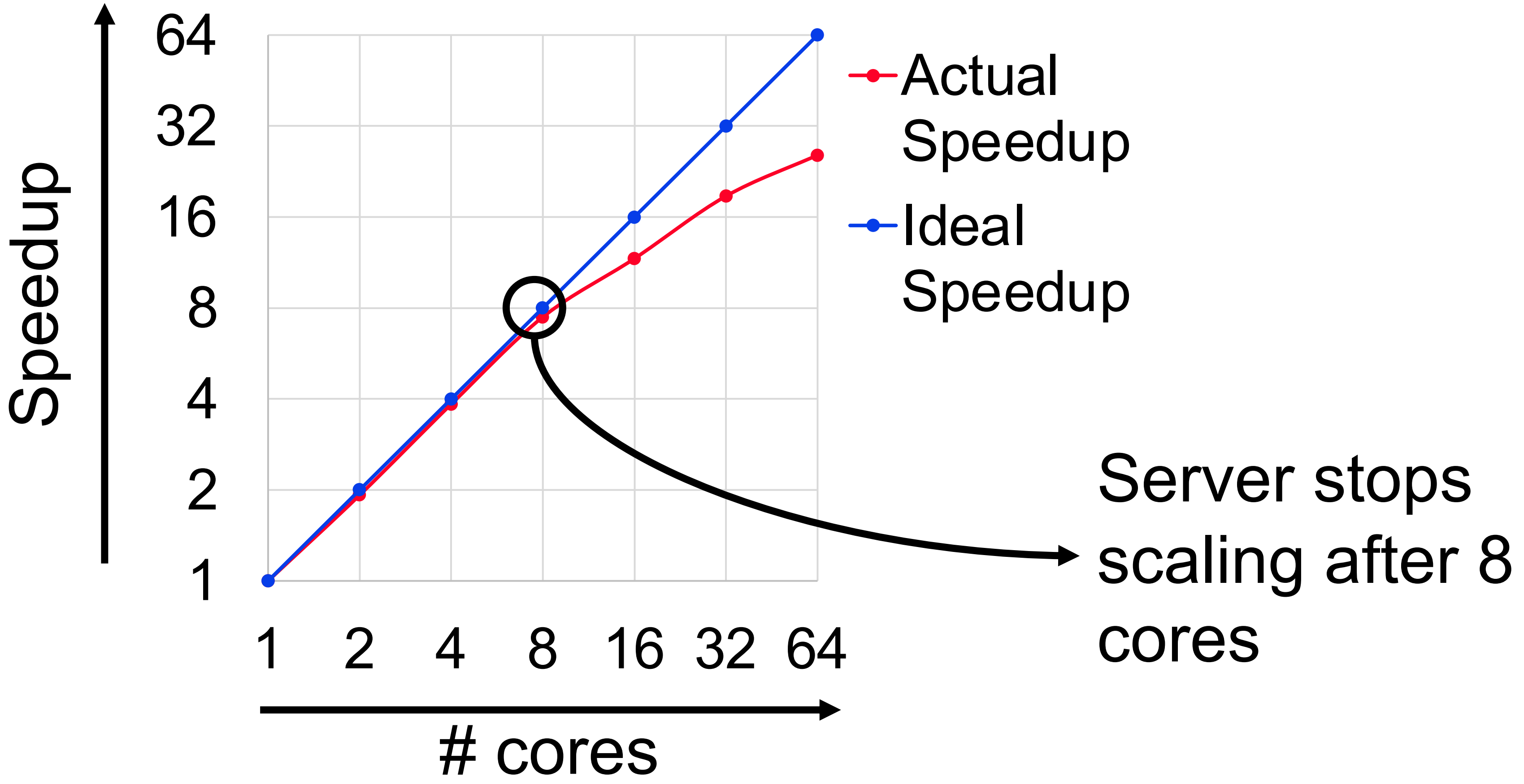


Example: A Real-World Server

- Memcached: one server keeps objects in memory for other servers
  - Each server runs on a separate machine
  - A few microseconds to fetch and object from another machine



Not All Monoliths Scale Well

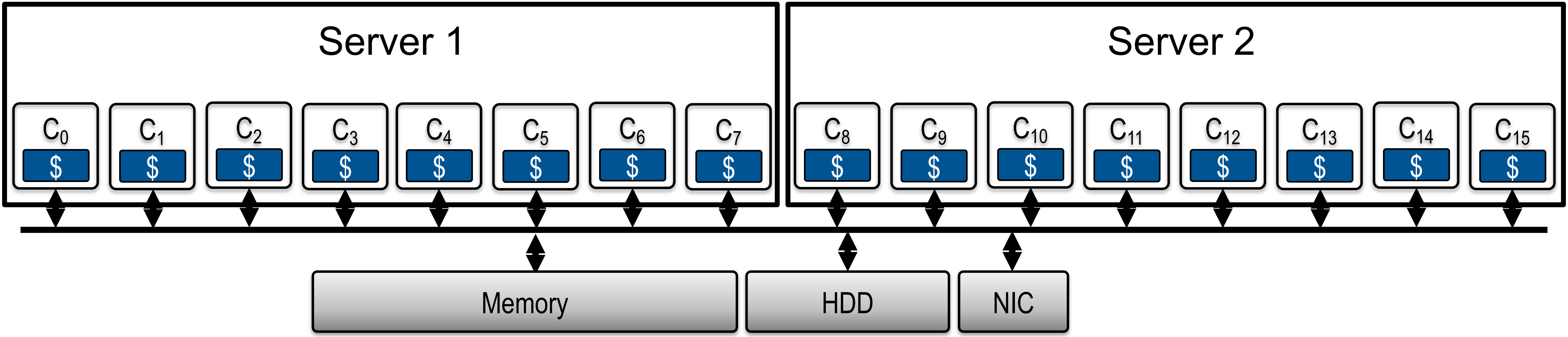


- Scaling the server beyond 8 cores does not utilize the cores effectively
  - Only 26x performance gain when run on 64 cores!

Running Multiple Instances of a Monolith

- A solution to use cores effectively is to run multiple monolith copies
  - Each instance (copy) behaves as an independent server
  - Threads among different instances do not synchronize with each other

- Each instance runs on a smaller number of cores and scales well
  - Both server 1 and server 2 can utilize eight cores effectively

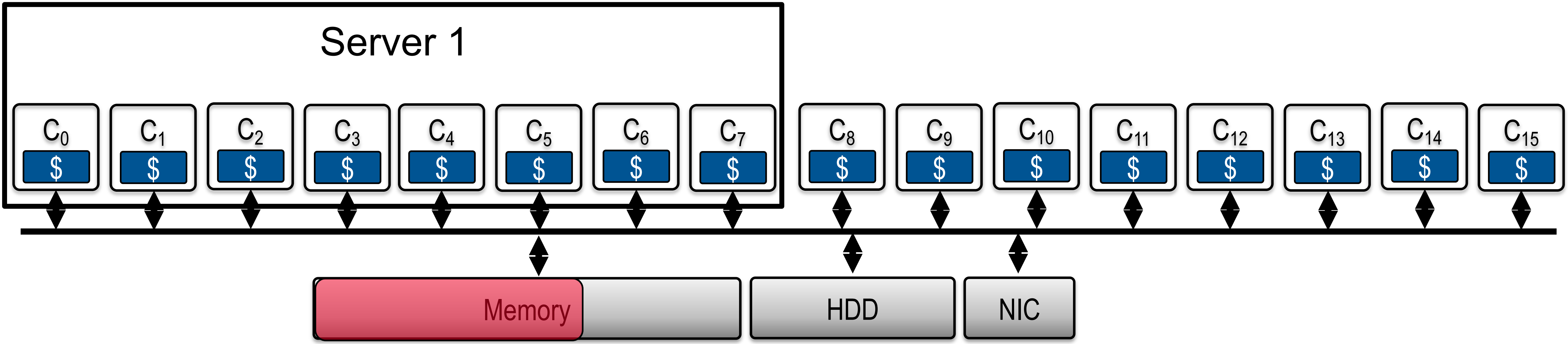


But, Monoliths Use Other Resources Too

- Memory capacity
  - Each monolith can take a big chunk of machine's memory
- Memory bandwidth
  - Analytic workloads (Spark) scale well but use a lot of memory bandwidth
- NIC bandwidth
  - Performance may be limited based on availability of NICs
- Accelerators
  - Only a few GPUs per machine needing only a few CPU cores to coordinate

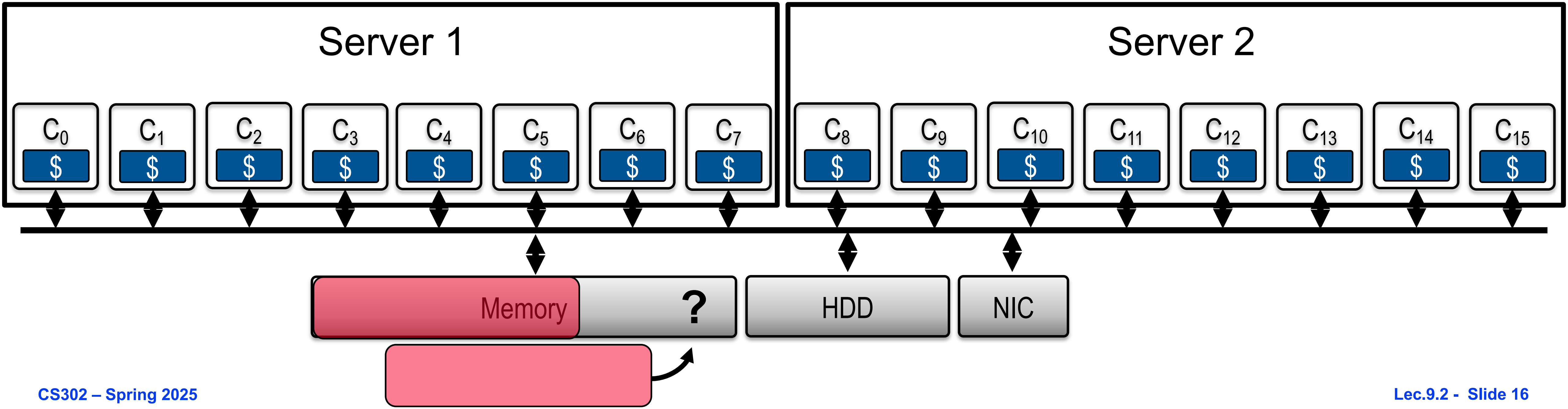
Example: Memory Capacity

- Each instance of a server can consume a large amount of memory
  - E.g., 60% of the system's total memory capacity is needed per instance



Example: Memory Capacity

- Each instance of a server can consume a large amount of memory
  - E.g., 60% of the system's total memory capacity is needed per instance
- Can not run many instances together on the same machine
  - System does not have enough memory to run the second instance



Monoliths Do Not Support Function Isolation

- Consider the case where one function is provided by a third party
  - Functions may come from various service providers
  - We may not even have direct access to the function
  - A monolith cannot run these four functions together

```
def save_file(req):
    ...

def get_line():
    ...

def handle_req(req):
    ...
```

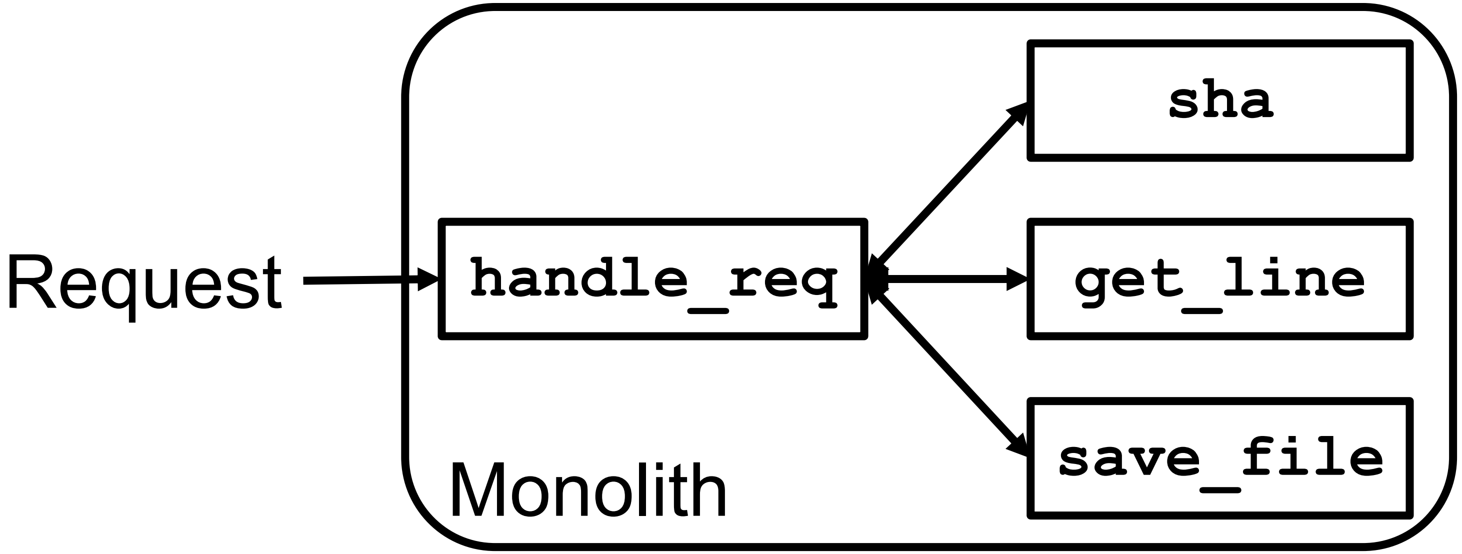
Our server

```
int sha(Request req):
    ...
```

RSA SECURITY

Functions Can Be Grouped

- Functions can be grouped based on the type of task they perform
- In our web server, a request can perform two distinct types of tasks
  - Save and serve files (**`save_file`** and **`get_line`**)
  - Perform hash computations (**`sha`**)



Idea: Split The Monolith

- These two tasks are independent and serve different needs
  1. Do these functions have to be in the same program?
  2. Do these functions have to execute on the same machine?

Idea: Split The Monolith

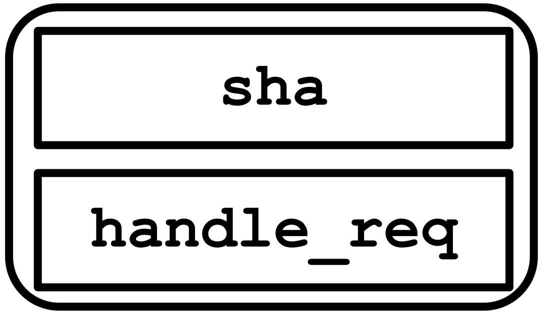
- These two tasks are independent and serve different needs
  1. Do these functions have to be in the same program?
  2. Do these functions have to execute on the same machine?

- No! These functions can be separated logically and physically

Microservices

- Groups of functions can be treated as separate "services"
  - Separated based on functionality and needs
  - Services have separate codebases, address spaces etc.
- This is called a Microservice architecture
  - A set of small (micro), independent and self-contained services

File service

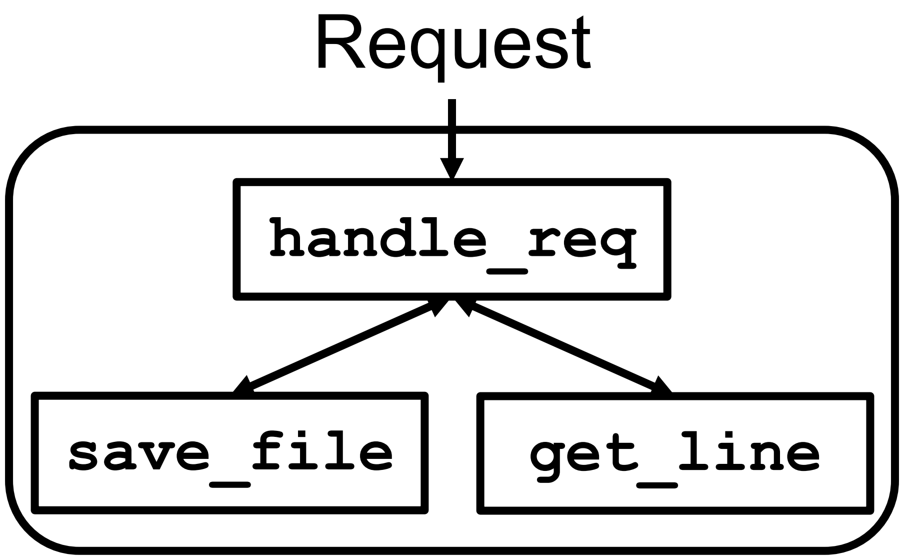


Hash service

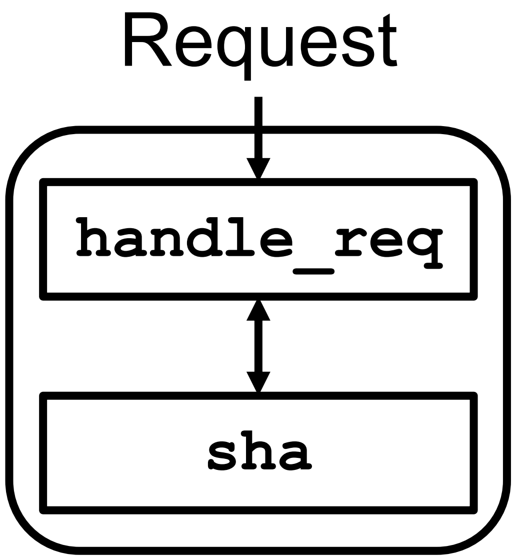
Microservices' Definition of Service

- A service executes a single task type in isolation
  - Each service has its own address to receive requests

- Each service has its own copy of **`handle_request`**
  - Processes incoming requests and chooses which function to execute



File service



Hash service

Microservices vs. Monoliths

Monolith	Microservices
One big program containing all functions	Functions are grouped into small programs (services)

server.py

```
def save_file(req):
    ...

def get_line():
    ...

def sha(req):
    ...

def handle_req(req):
    ...
```

file.py (File service)

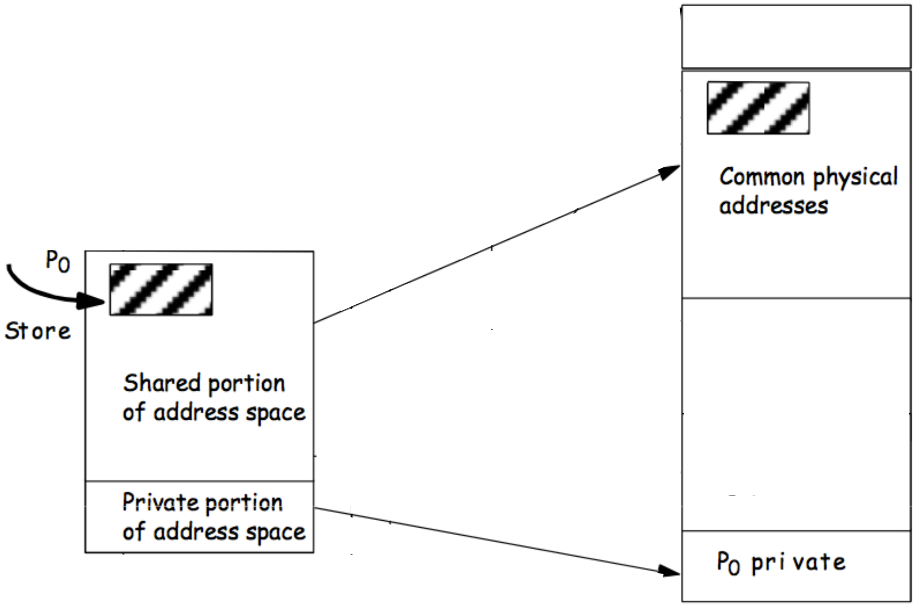
```
def save_file(req):
    ...

def get_line():
    ...

def handle_req(req):
    ...
```

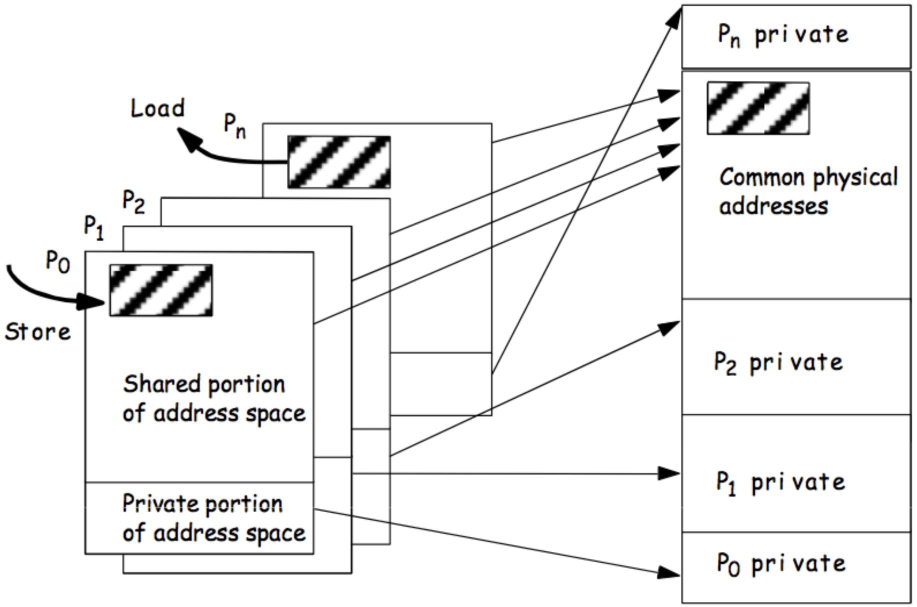
hash.py (Hash service)

```
def sha(req):
    ...

def handle_req(req):
    ...
```

Microservices vs. Monoliths

Monolith	Microservices
One big program containing all functions	Functions are grouped into small programs (services)
All functions run in the same address space	Each service runs in its own address space





Microservices vs. Monoliths

Monolith	Microservices
One big program containing all functions	Functions are grouped into small programs (services)
All functions run in the same address space	Each service runs in its own address space
Functions call other functions and use shared memory to communicate	Functions in various services can also “call” each other and communicate using messages (To be covered in detail later!)

Microservices Scale Better Than Monoliths

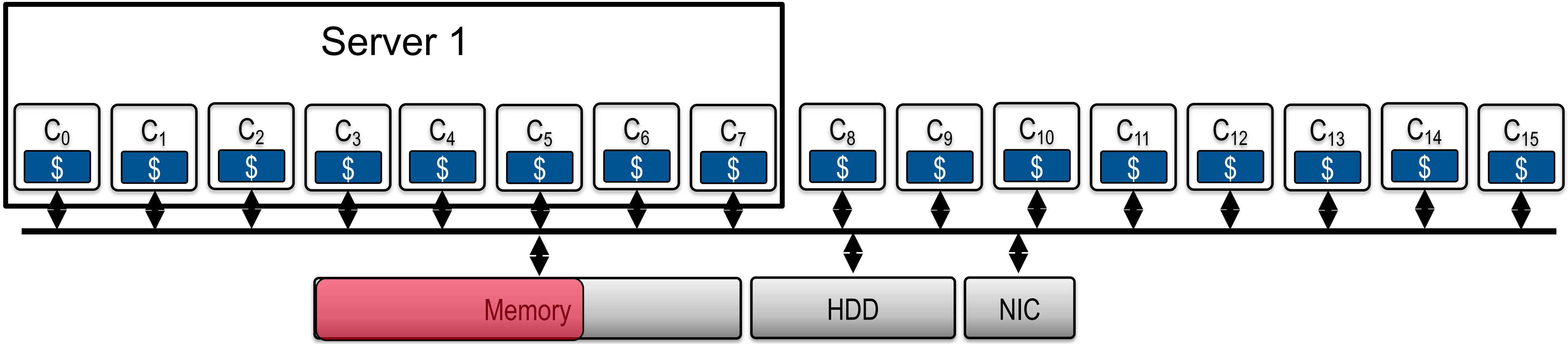
- Services have various machine requirements
  - **`file.py`** uses more memory
  - **`hash.py`** uses more compute

- Services are not used uniformly
  - Some services are more popular than others
  - These services handle more requests than others

- Each service can be scaled independently of the other services!
  - Popular services can be scaled without scaling the other services
  - Resources can be better allocated depending each service’s needs

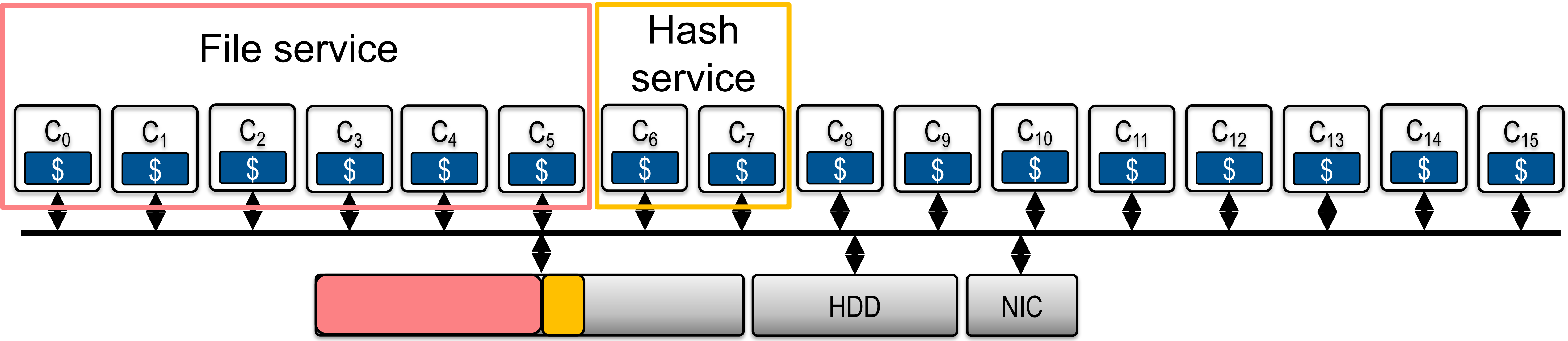
Microservices Scale Better Than Monoliths

- Previously when using a monolith, one instance of a server used:
  - Eight cores and 60% of the system's total memory



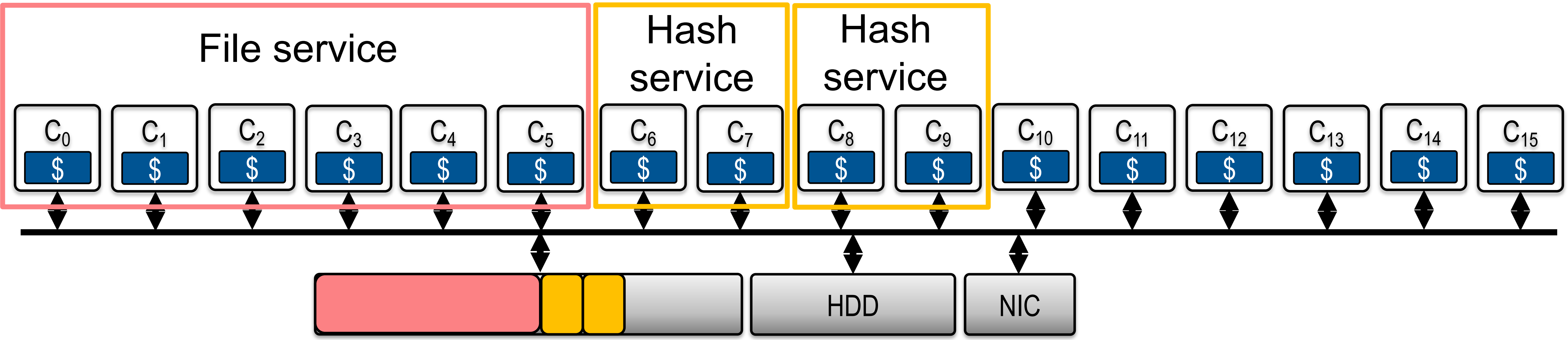
Microservices Scale Better Than Monoliths (Slide 28)

- After splitting the monolith into two services, assume one instance of:
  - File service runs on six cores and uses 50% of the system's total memory
  - Hash service runs on two cores and uses 10% of the system's total memory



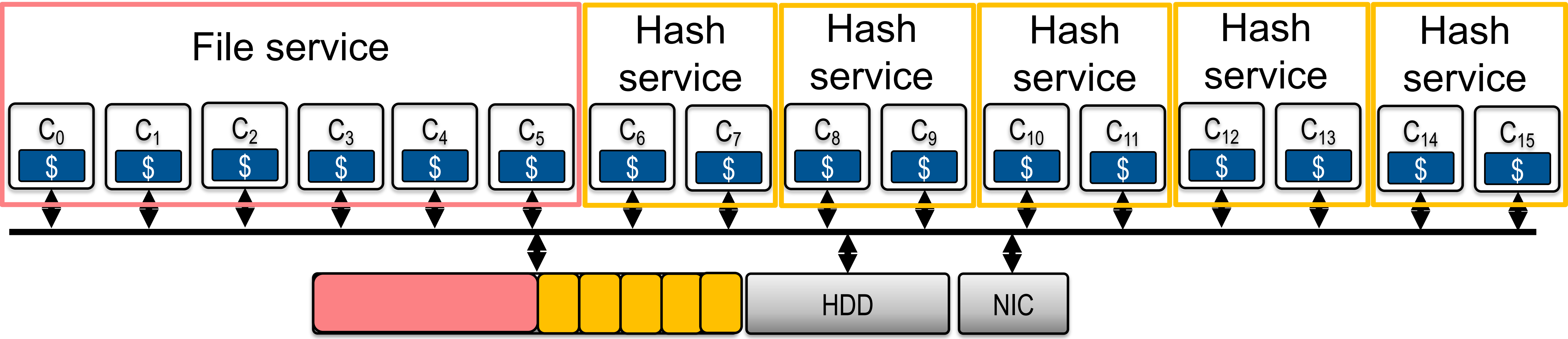
Microservices Scale Better Than Monoliths (Slide 29)

- What if 2x more requests/second started arriving for Hash?
- Hash service can be scaled independently!
  - Another instance of hash service gets created
  - Only two more cores and 10% more memory needs to be allocated
  - The instance of file service remains unchanged



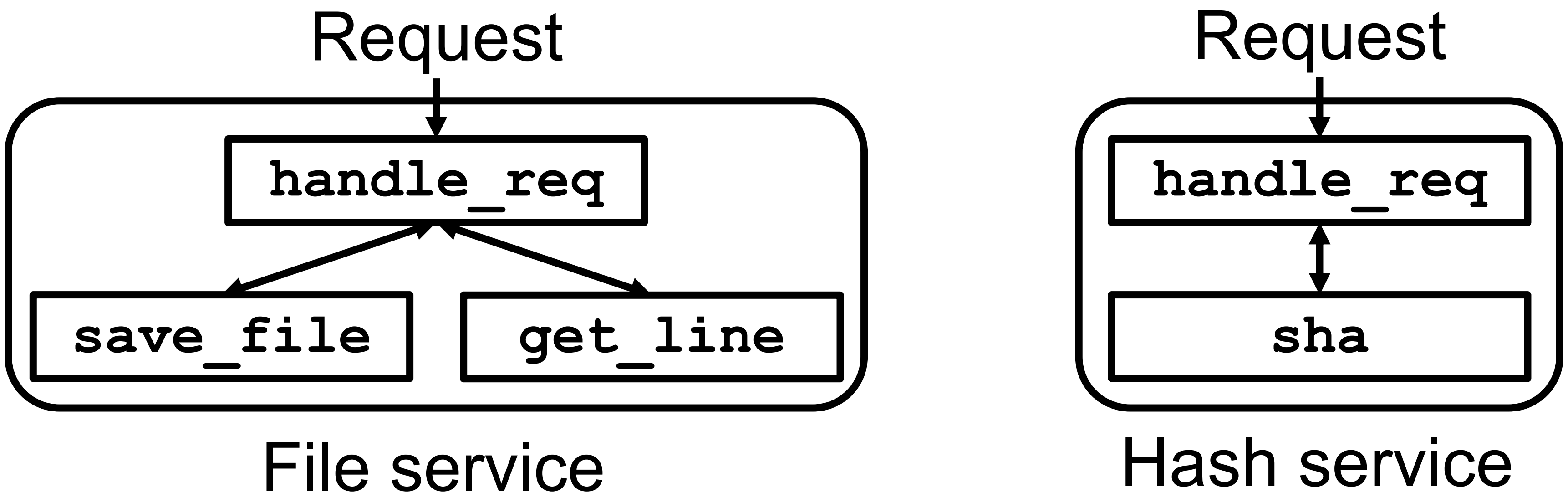
Microservices Scale Better Than Monoliths (Slide 30)

- This was not possible when using a monolith
  - A new instance of the entire server must have been created
- Microservices offer flexibility on how to scale services as needed!
  - Five instances of hash service can be created to serve requests
  - Completely utilizes all cores and memory



Microservices Can Run On Different Machines

- Diverse requirements for file service and hash service
  - File service needs a machine with more memory and cheap storage
  - Hash service needs a machine with less memory and no storage



- Services may run independently
  - Each service can run on a machine that matches its requirements

Microservices Can Run On Different Machines

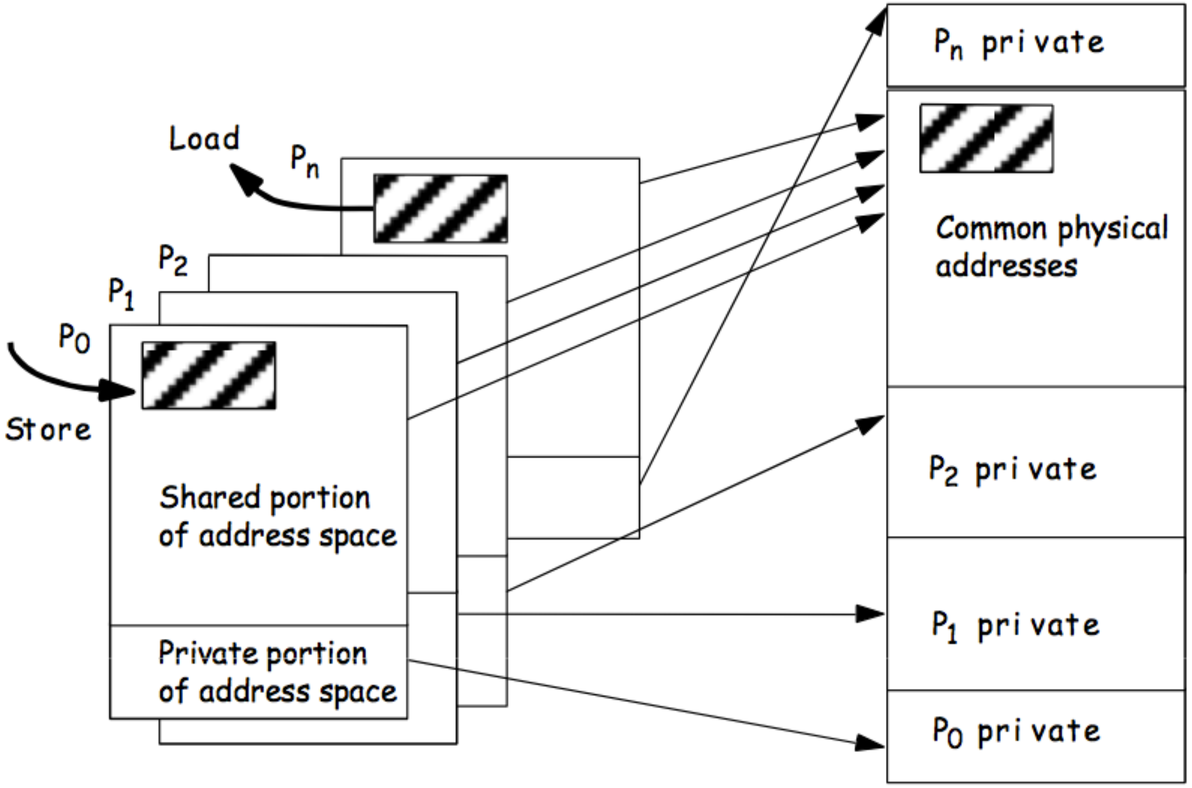
- Cloud providers offer servers with a variety of resources
  - Cheaper plans can be bought depending on the service!

Instance name	On-Demand hourly rate	vCPU	Memory	Storage	Network performance
t4g.nano	$0.0042	2	0.5 GiB	EBS Only	Up to 5 Gigabit
t4g.micro	$0.0084	2	1 GiB	EBS Only	Up to 5 Gigabit
t4g.small	$0.0168	2	2 GiB	EBS Only	Up to 5 Gigabit

Example Amazon AWS offerings

Microservices Allow Isolation Among Services

- Each service runs in its own address space
  - A service cannot directly read or write another service's private address space
  - Allows isolation across various services



Microservices Allow Flexibility for Developers

- Services can be written independently of each other
  - Choose the best programming language according to a service's needs
  - E.g., Python for ML services, C/C++ for data services, JS for web services., etc.

```
def save_file(req):
    ...

def get_line():
    ...

def handle_req(req):
    ...
```

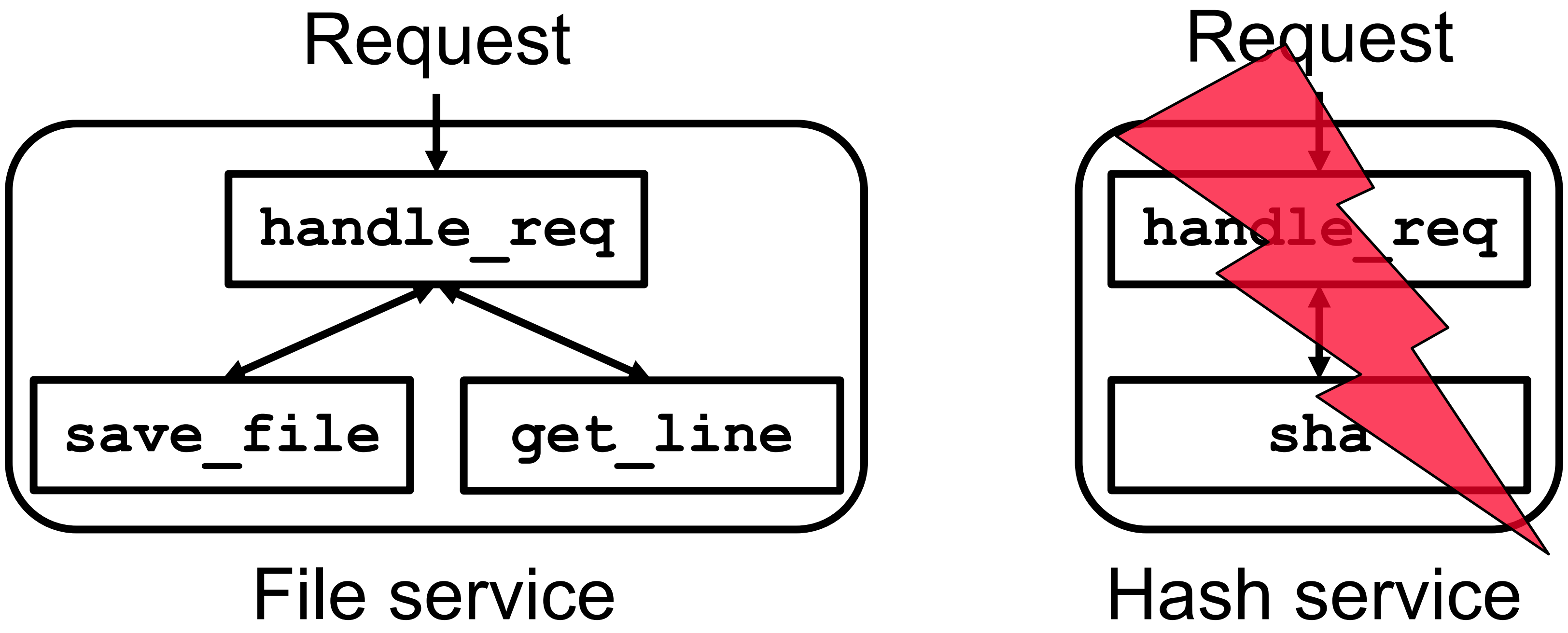
file.py (File service)

```
char * sha(Request req):
    ...

void handle_req(Request req):
    ...
```

hash.cpp (Hash service)

Microservices Allow Fault Tolerance

- One service failing does not lead to the entire workload failing
  - Services are isolated with address spaces
  - In contrast, a single error in a monolith crashes the entire monolith



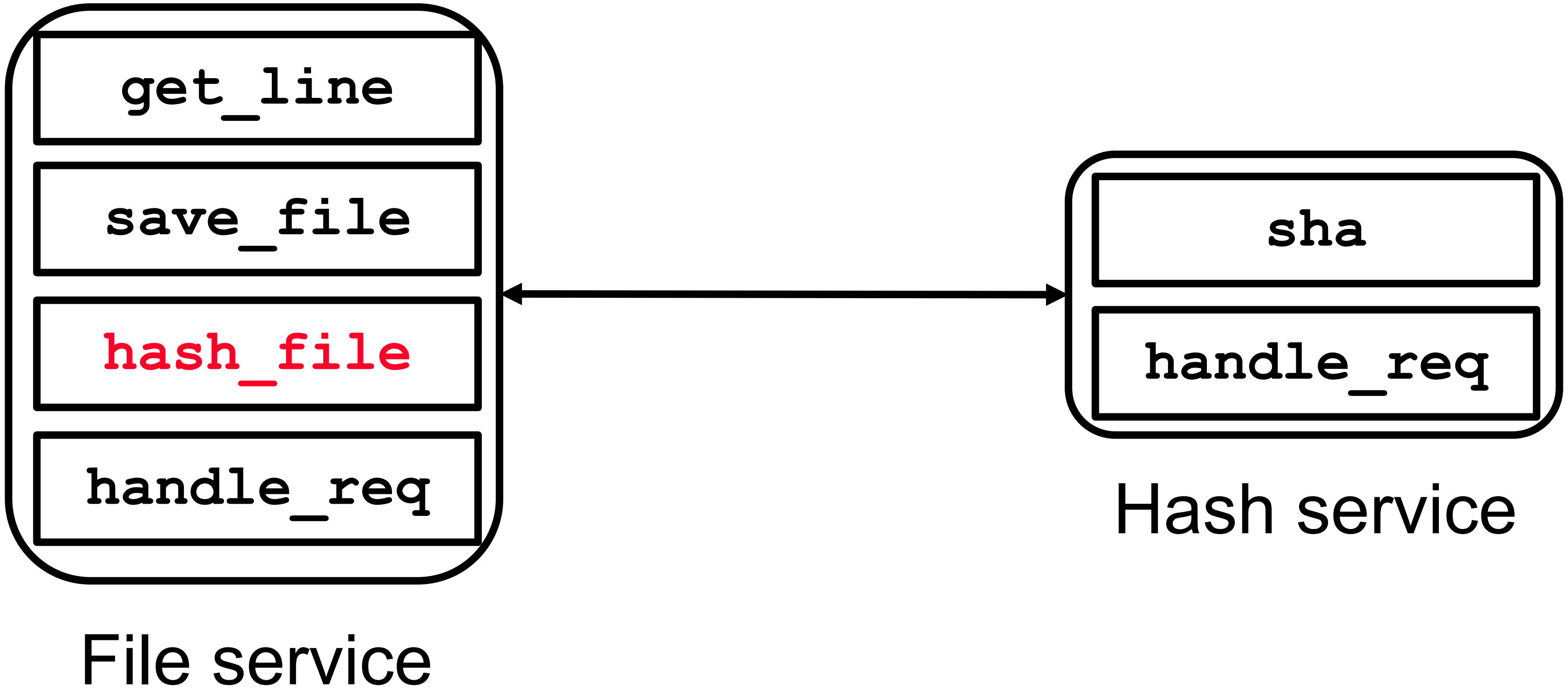
Microservices Allow Reusability

- Services are isolated but can be accessed from any other service
  - Enables reusability by other users and applications
- E.g., consider we want to add a hash file function to file service
  - Reads a line from a file and computes the hash of the line
  - It should be able to reuse the functions from the hash service

```
def hash_file():
    content = '\n'.join([line for line in get_line])
    hash = sha(content)
    return hash
```

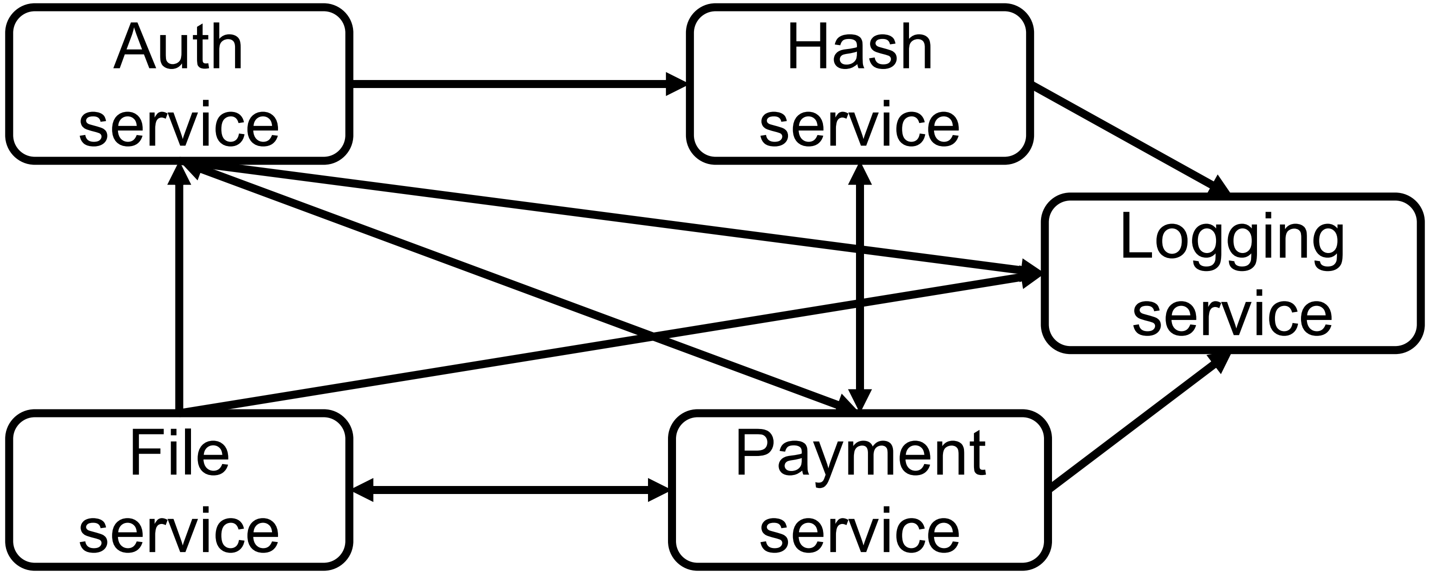
Connecting Services

- Services need to be able to interact with each other
  - Functions in different services should be able to call each other
  - Communication across different services can happen through messages

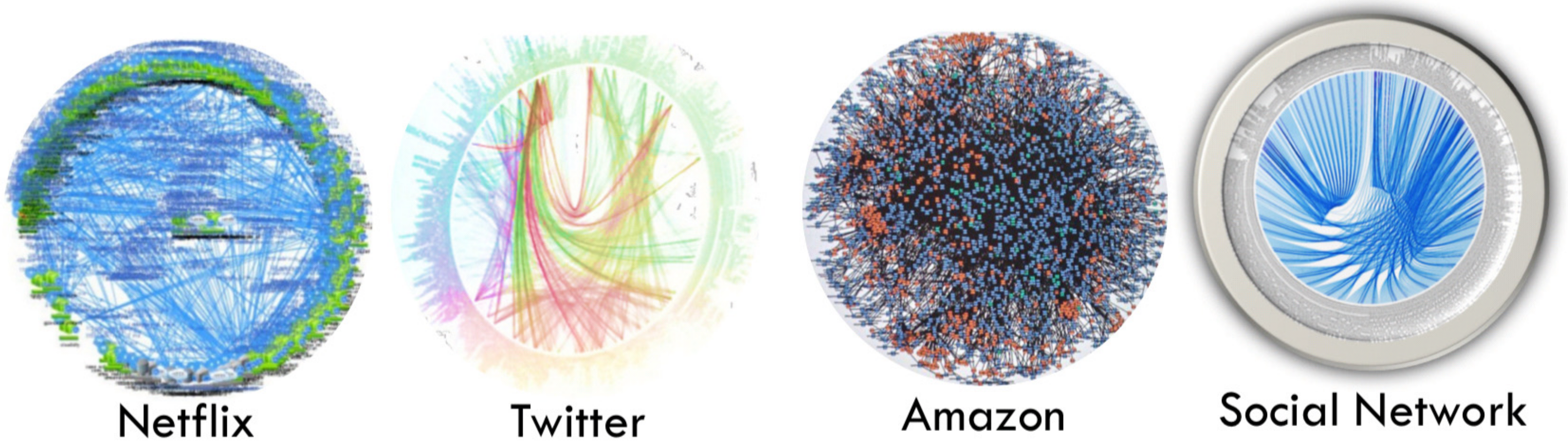


Interaction Among Services

- Communication becomes critical as the number of services increases
  - E.g., consider adding authentication, payment and data logging services
- The connectivity graph is also important in scalability
  - $O(N^2)$ connections possible where N is the number of services
  - Services are also from diverse programming languages



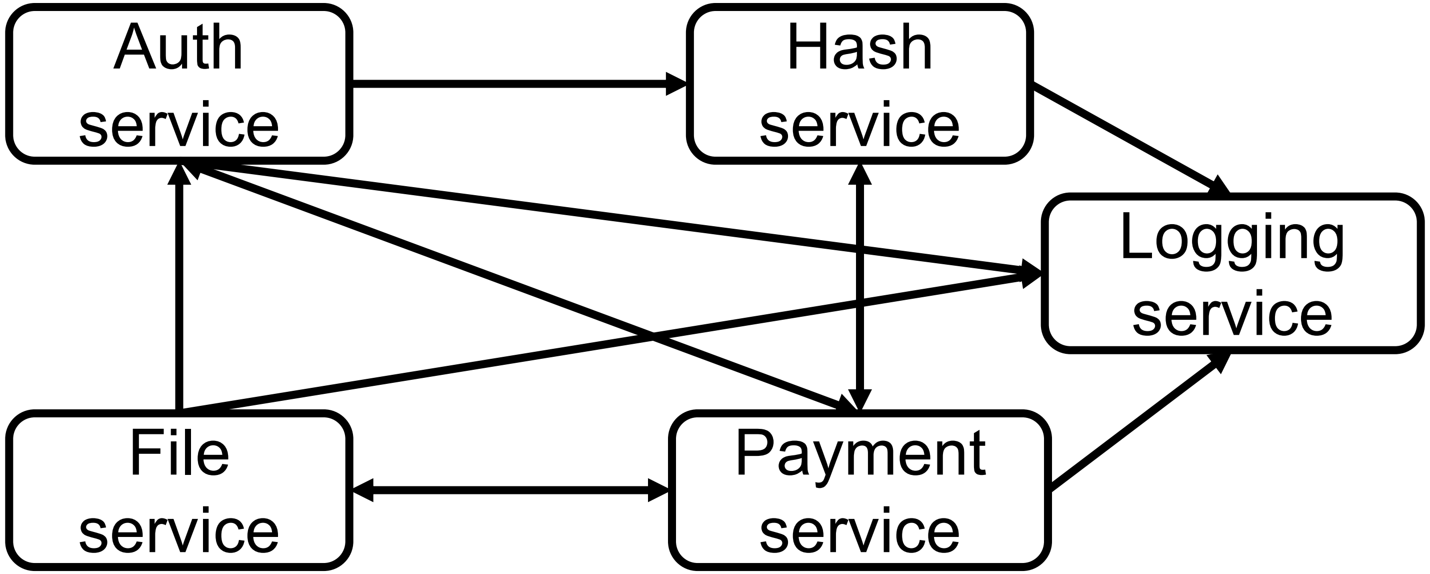
Example: Real-World Connectivity Graphs



[source: DeathStarBench, ASPLOS’19]

Need a Standard Communication Protocol

- Must be fast (faster than the runtime of each service)
- Must be fault-tolerant
- Able to find each service (in the network)
- Able to pass input to the service and receive its output



RPC: A Remote Procedure Call

- Microservices communicate with function calls
  - Local calls: Calls by a function to another function within the same service
    - This is our conventional function call (e.g., in Python, C)
    - We call this **local procedure call** (LPC)
  - Remote calls: Calls by a function to another function in a different service
    - This is new and used for microservices
    - A **remote procedure call** (RPC)
- Procedure (Function) calls can be abstracted so both local and remote calls look the same for our program

RPC : A Remote Procedure Call

- When a program calls a procedure to execute in another address space as if it was a local procedure, it is called a remote procedure call or an RPC

- We will use RPC to run our two services separately
  - Our File service in Python
  - Our Hash service in C++
    - Better suited for real-time and compute heavy applications

RPC : A Remote Procedure Call

- A reminder of our file service using the hash service using LPC
- What steps are involved in a LPC and how are they changed?

```
def sha(req):
    hash_arr[req.id] = calc_hash(req.payload)
    return


def hash_file(req):
   ... // get's file and prepares it for sha
   sha(req)
```

RPC: Function Definition

- First step: Function definition
- This is where we define the function signatures as interfaces
- Then we implement the functions

RPC: Function Definition

- **`hash_file`** depends on **`sha`**, so we need to define the **`sha`** function first

```
def sha(req) -> str:
    hash_arr[req.id] = calc_hash(req.payload)
    return

def hash_file():
    content = '\n'.join([line for line in get_line])
    hash = sha(content)
    return hash
```

RPC: Function Definition

- What is involved in defining a function?
  - Function name
  - Input
  - Output

```
def sha(req) -> str:
        ...
```

RPC: Function Definition

- In LPC (a regular function call) we just define the function in code
  - Can be just the function interface or header
- Importing and referencing the function by other parts of the code is handled by the compiler/interpreter
- How can we do this if the two functions are in two different address spaces and may even be in two different languages?

Interface Definition Language

- Use an **interface definition language** (IDL) to describe functions
  - This language will not run any code
  - Common format to describe functions across languages
- First define each of our services

```
service hash{
        ...
}
```

RPC: Function Definition

- Define the functions of one microservice exposed to other services
  - This is just an interface and has no implementation

```
service hash{
      rpc sha(Request) returns (StringValue) {}
}
```

RPC: Function Definition

- RPC library will take in the IDL code as input
  - Example: Protobuf (used by Google), Thrift (used by Facebook)
- RPC library has implementation for various languages like C, Python, etc.
- RPC library will generate code in the language of all our services from one interface definition code

RPC: Server Stub

- The code generated for the service that implements and serves a function is called the server stub

```
// generated by rpc library in C for the hash service
class HashService::service{
     // Abstract function, later implemented by developer
     virtual char * sha(Request req:)= 0;
}
```

RPC: Server Stub

- RPC leaves the function as an abstract function
- The developer implements it

```
// generated by rpc library in C for the hash service
class HashService::service{
      // Abstract function, later implemented by developer
      virtual char * sha(Request req:)= 0;
}
```

RPC: Client Stub

- The code generated for the services that use a function is called the client stub

```
# generated by rpc library in Python for the file service
class HashService:
     def sha(req: Request)-> String:
          # RPC sending the request to hash service
          resp = sendReq(req)
          ...
```

RPC: Function Definition

- Client stub is a concreate code
- RPC implements sending the request to the service that serves it

```
# generated by rpc library in Python for the file service
class HashService:
     def sha(req: Request)-> String:
          # RPC sending the request to hash service
          resp = sendReq(req)
          ...
```

Summary

- Microservices allow
  - Isolation of services
  - Scalability of services based on their needs
  - Fault isolation
  - Heterogenous tech stack
  - Reusability

- Communication becomes complex with more services
- RPC abstracts communication among services as normal function calls
  - To be continued in the next lecture!