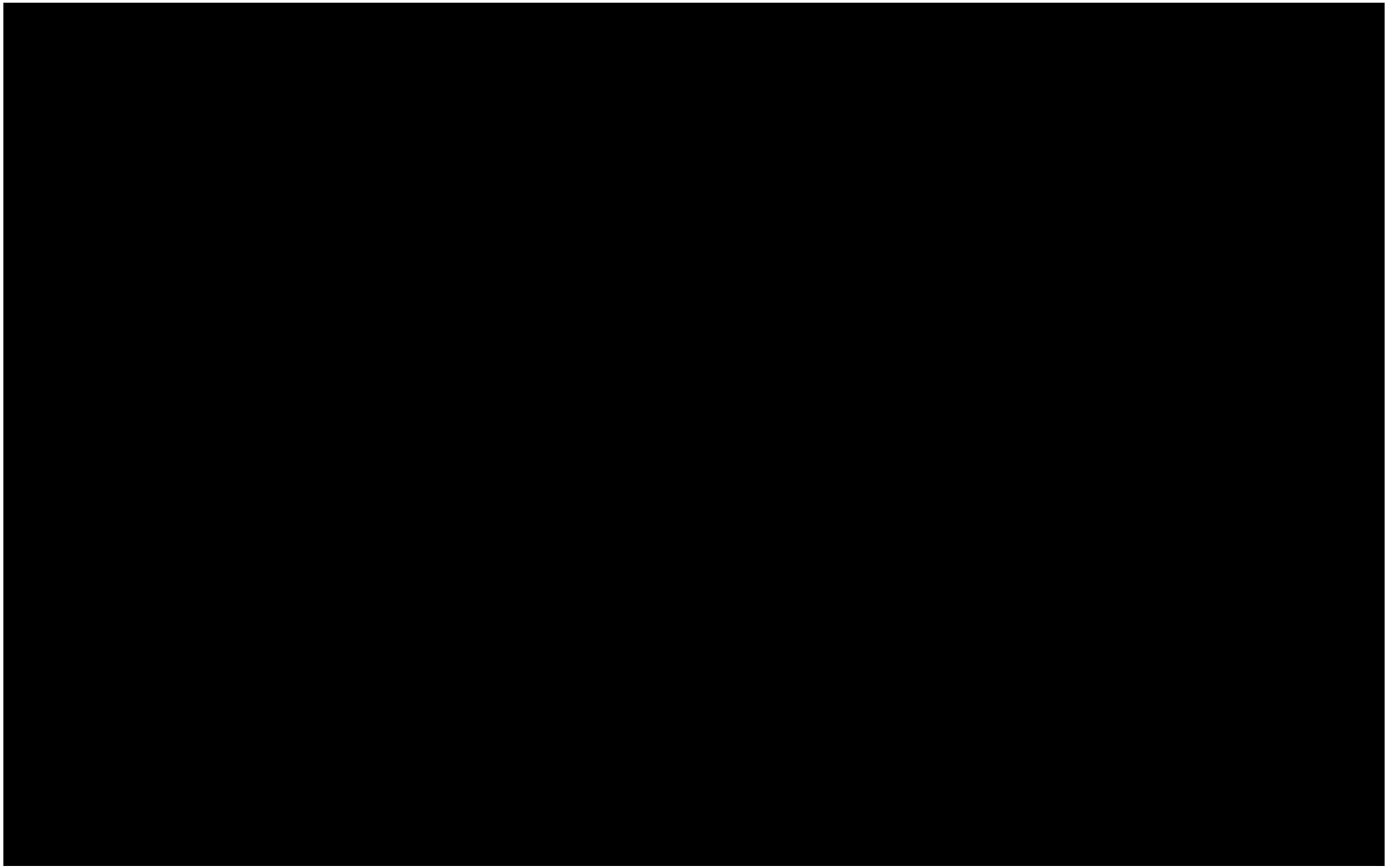
Cocoa Interprocess Communication with XPC

Session 241

Tony Parker
Software Engineer, Cocoa Frameworks

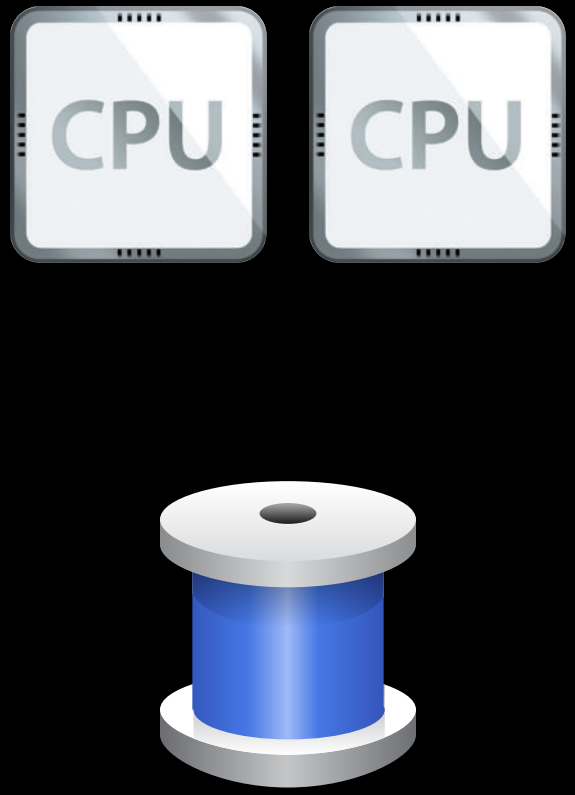


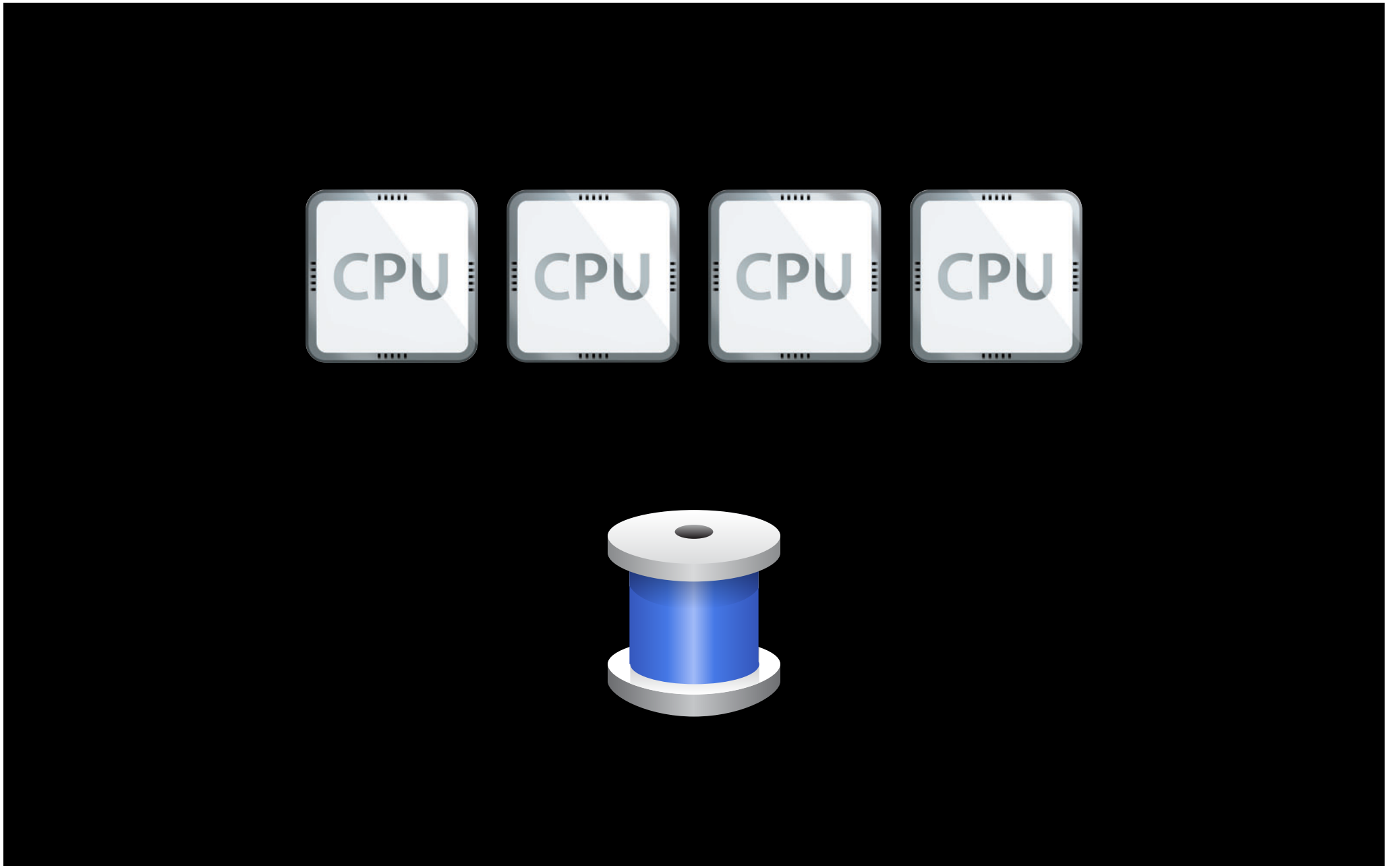
First

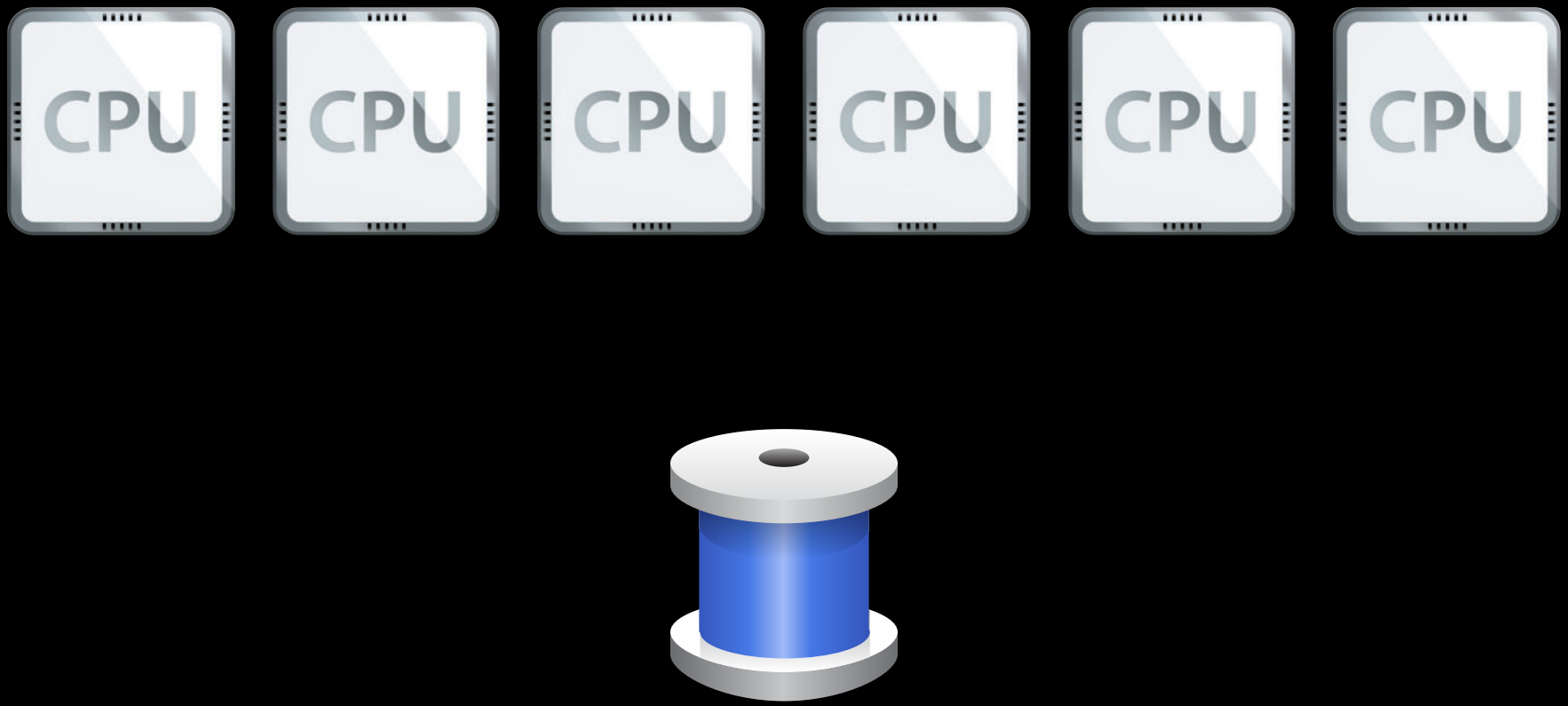
a quick trip back in time

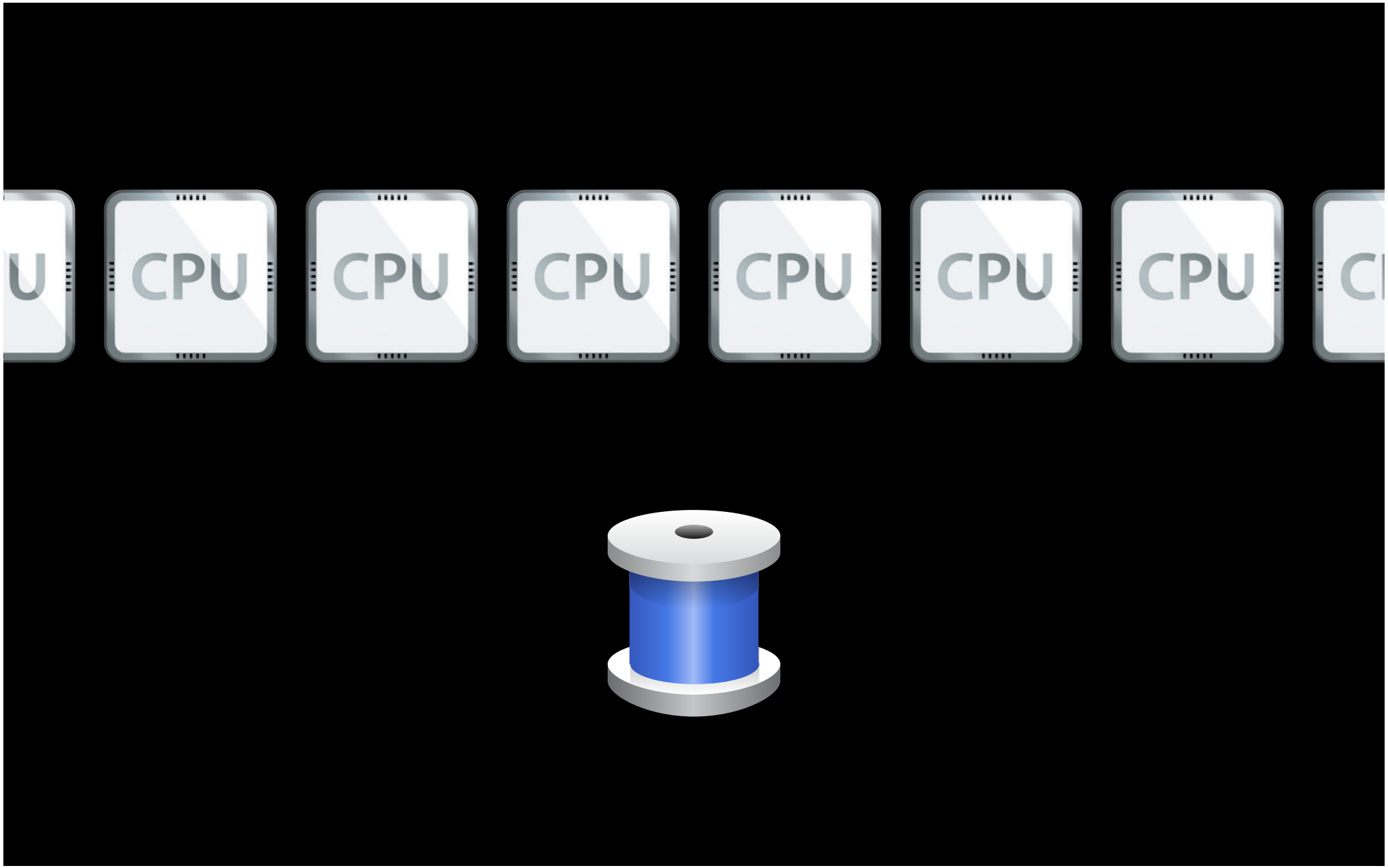

CPU

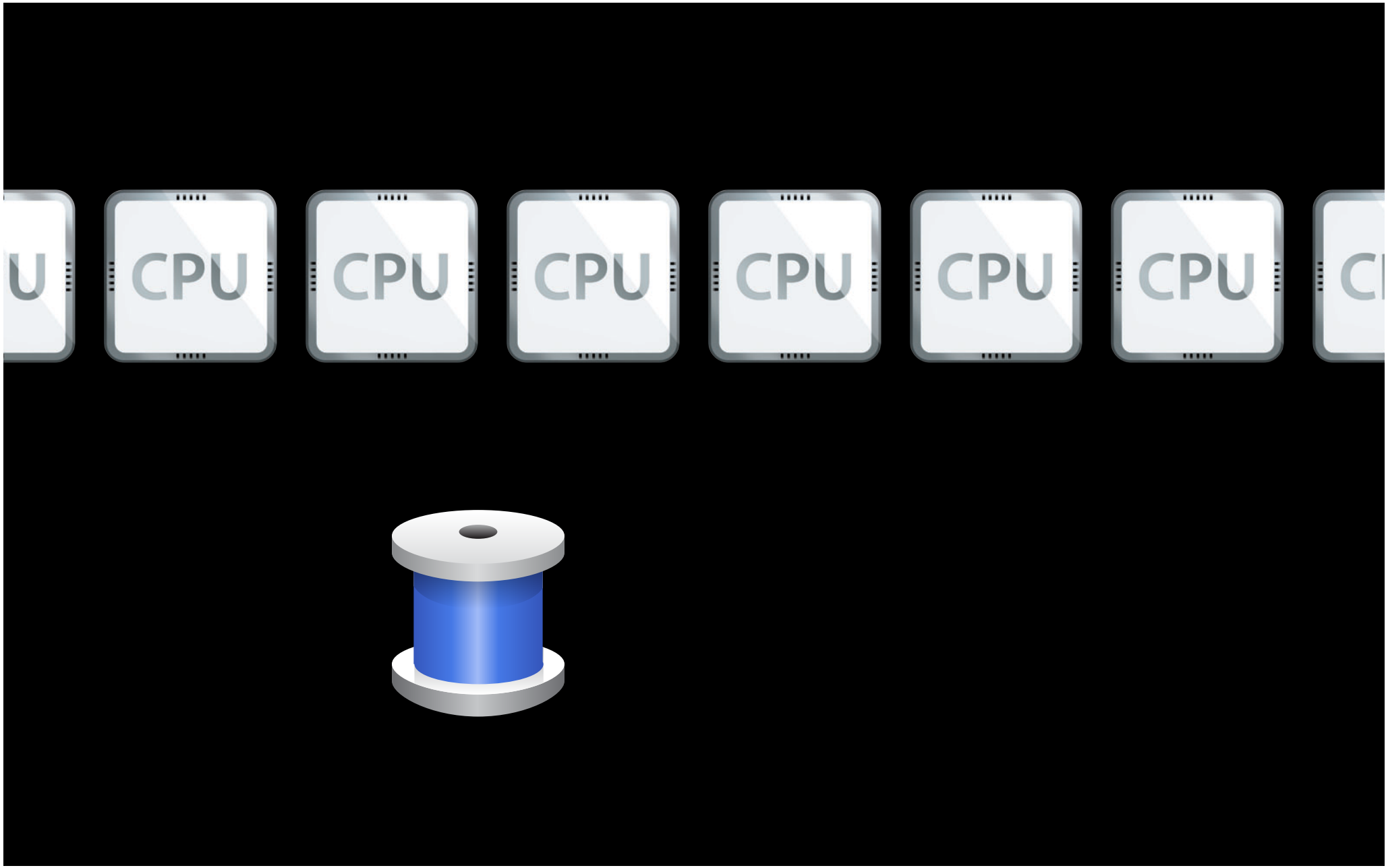

CPU

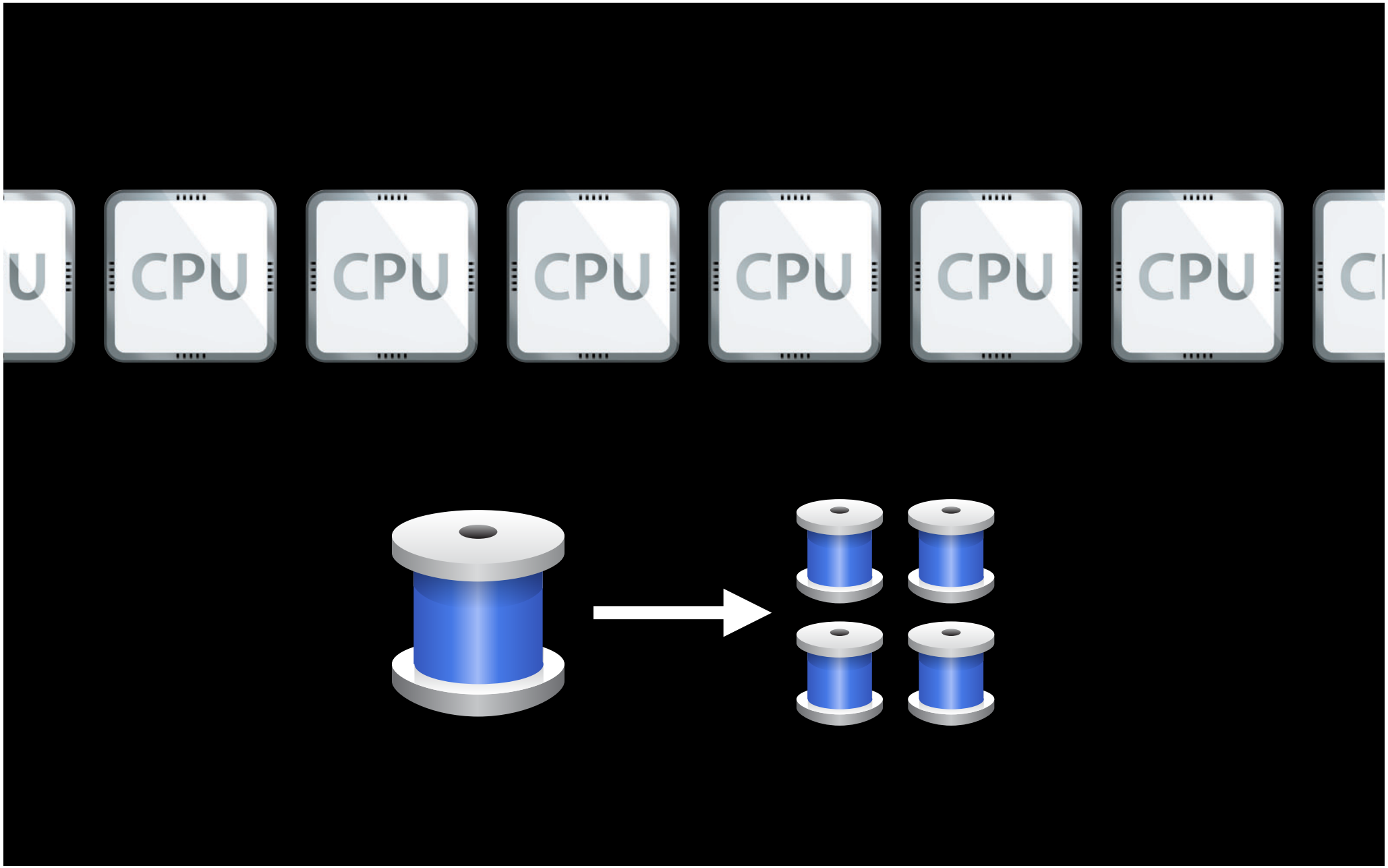

CPU
CPU

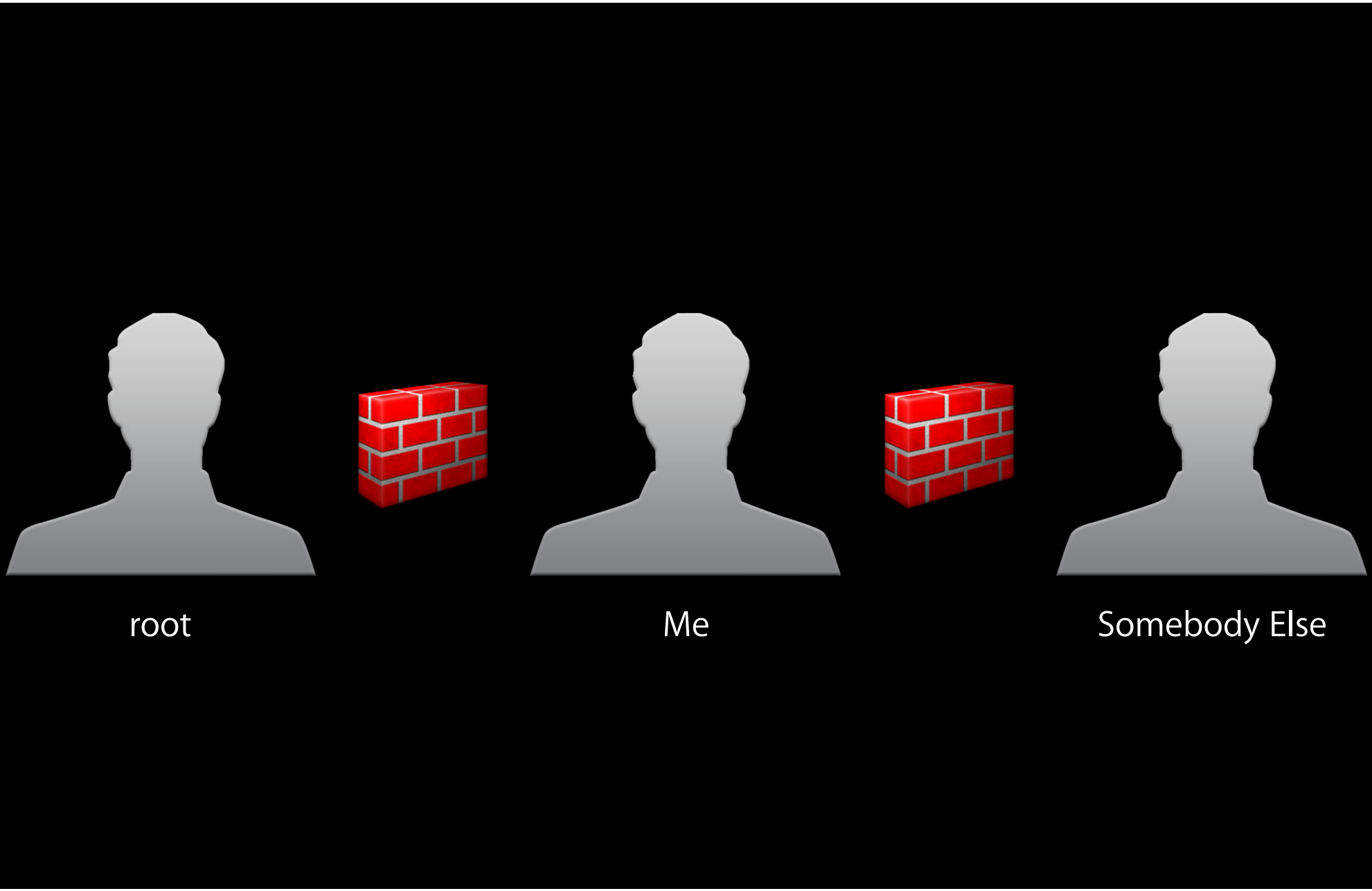

CPU
CPU
CPU
CPU

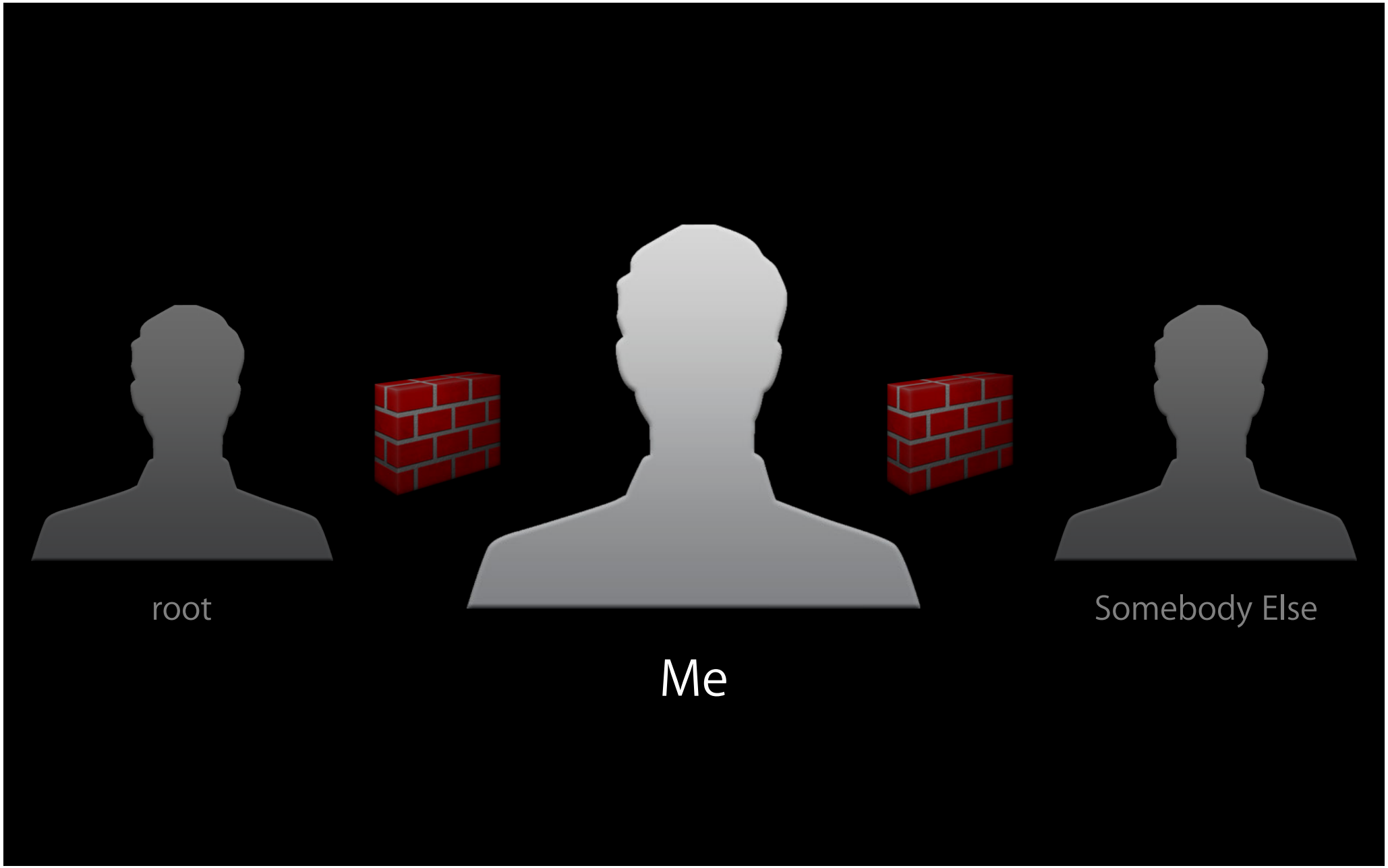

CPU
CPU
CPU
CPU
CPU
CPU

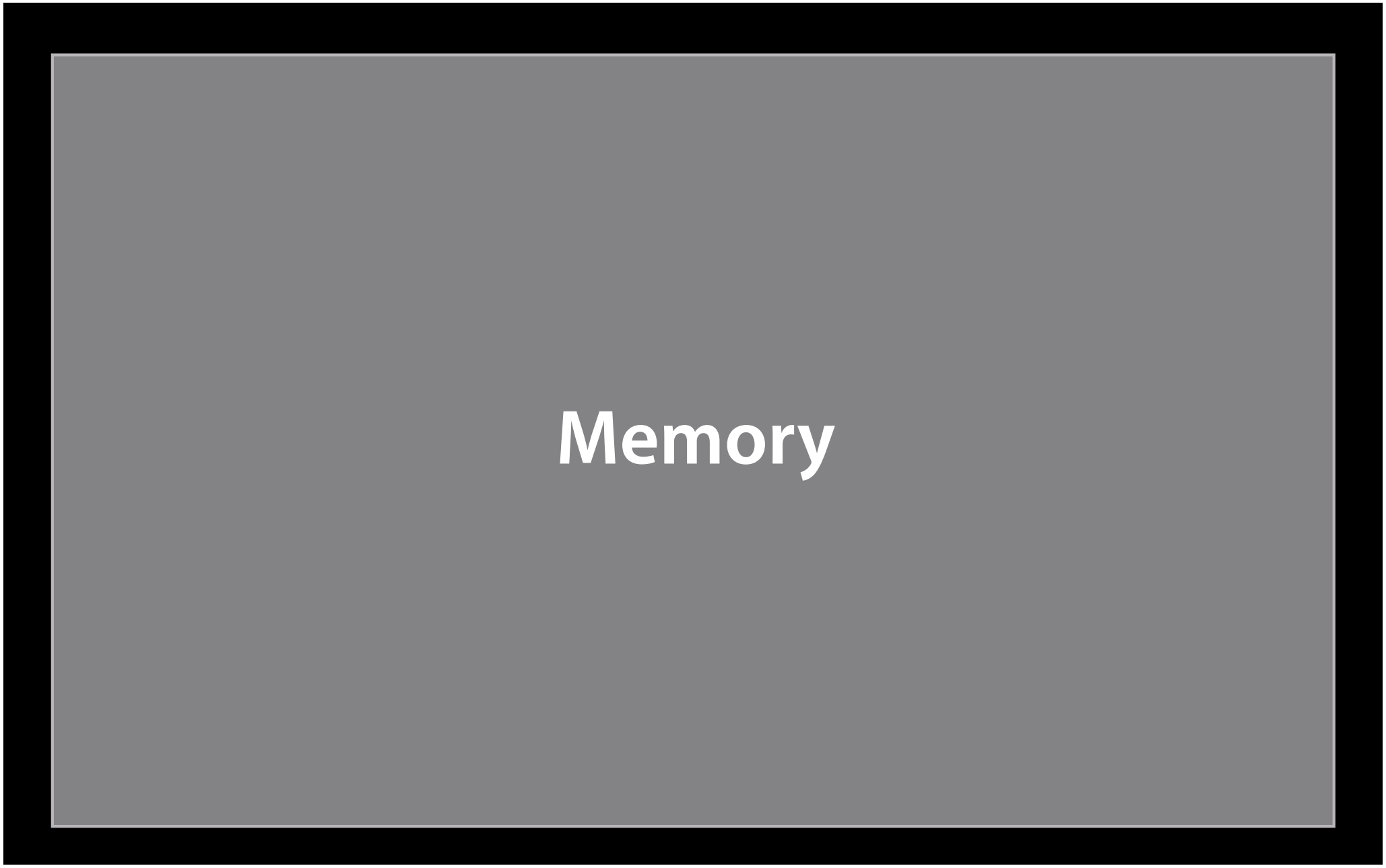

CPU
CPU
CPU
CPU
CPU
CPU
CPU

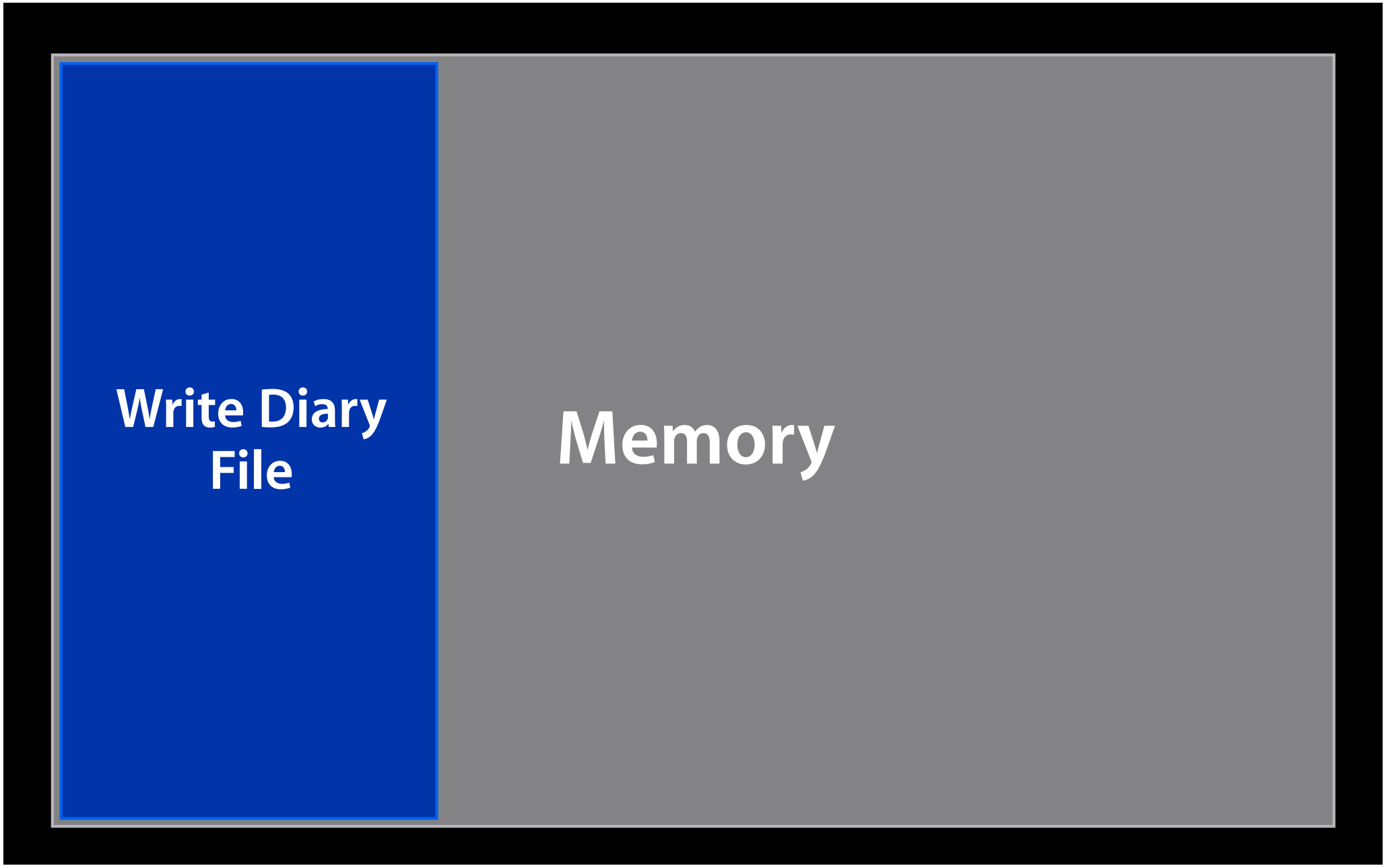

U
CPU
CPU
CPU
CPU
CPU
CPU
C

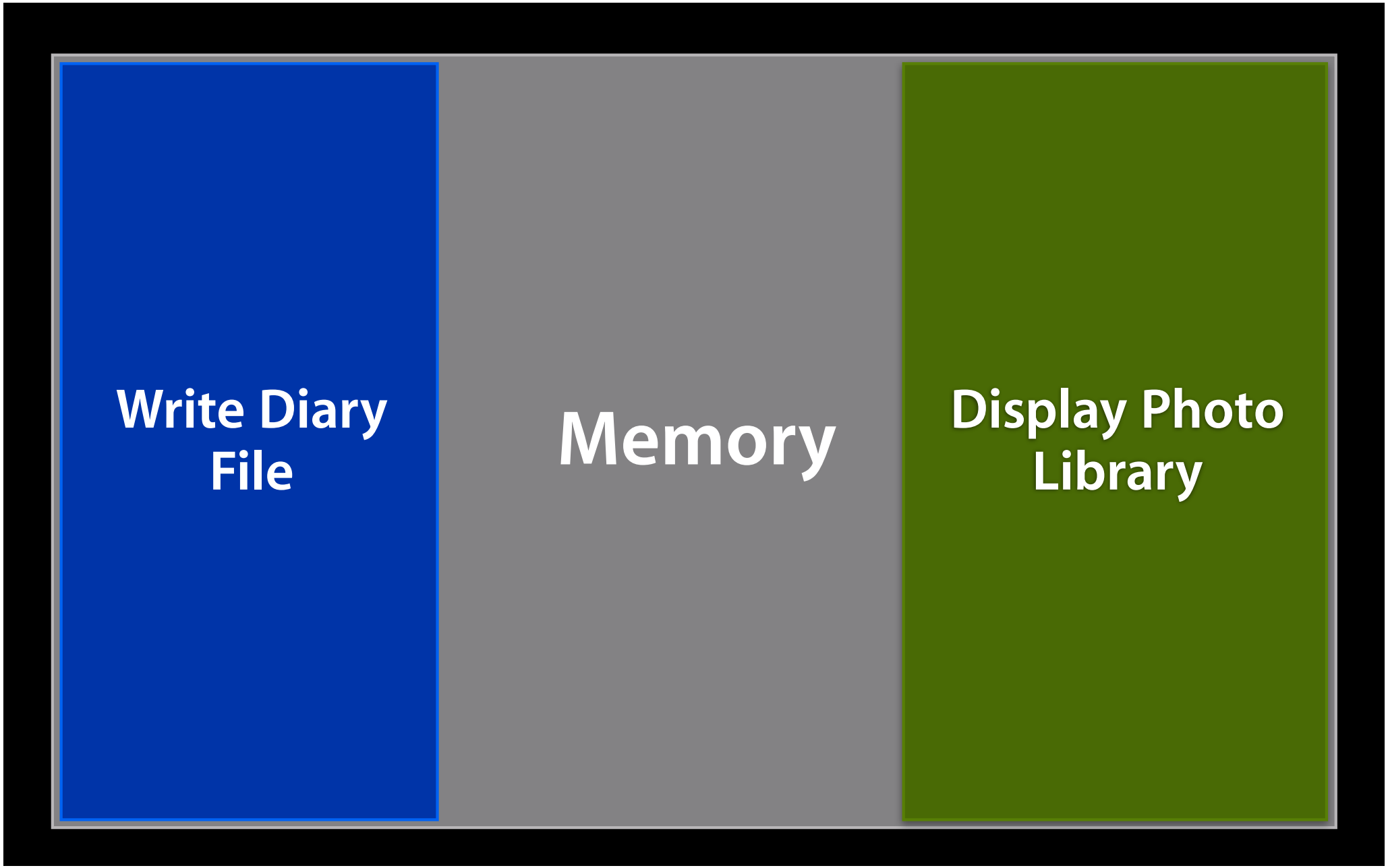

CPU
CPU
CPU
CPU
CPU
CPU
CPU

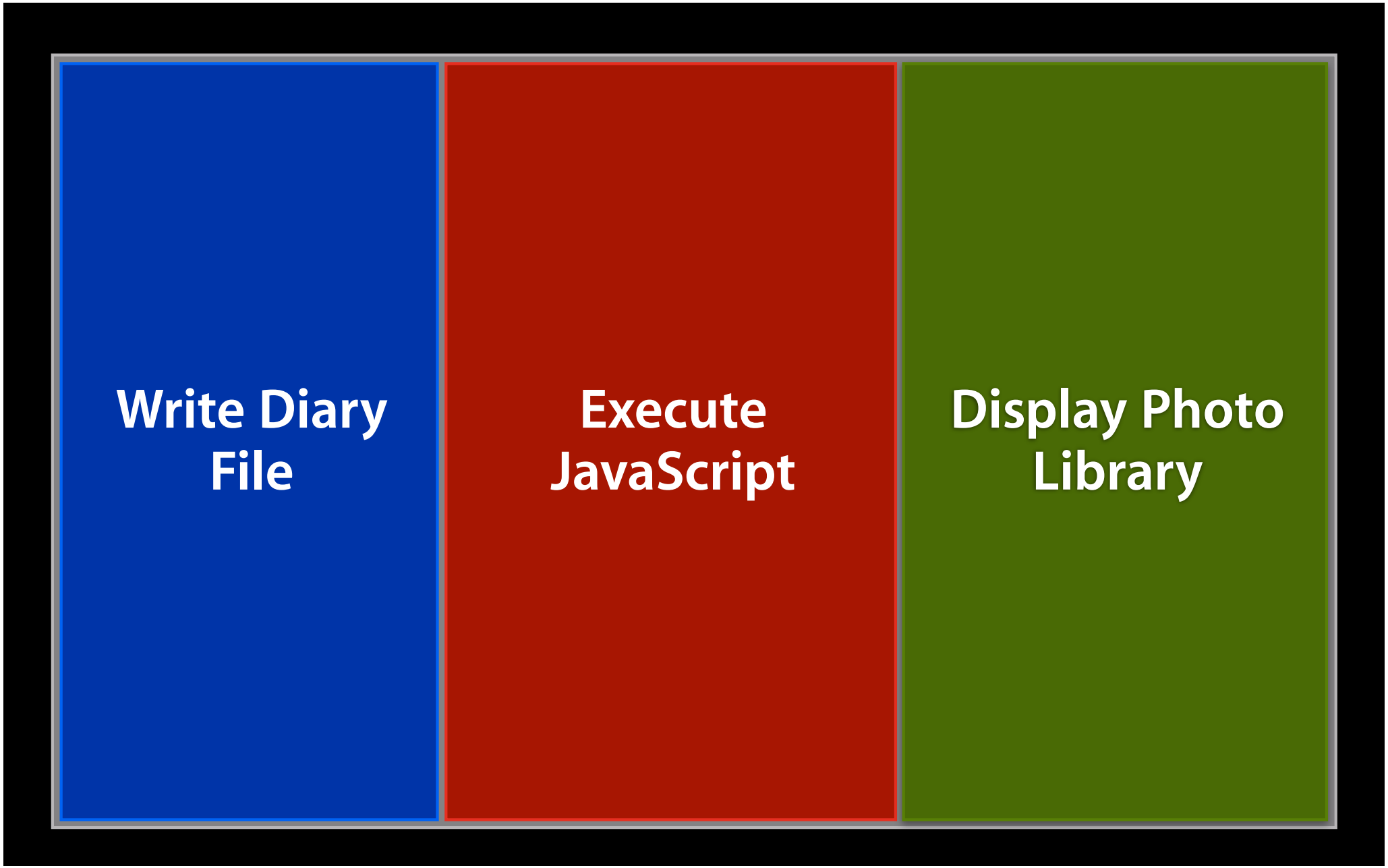

root
Me
Somebody Else

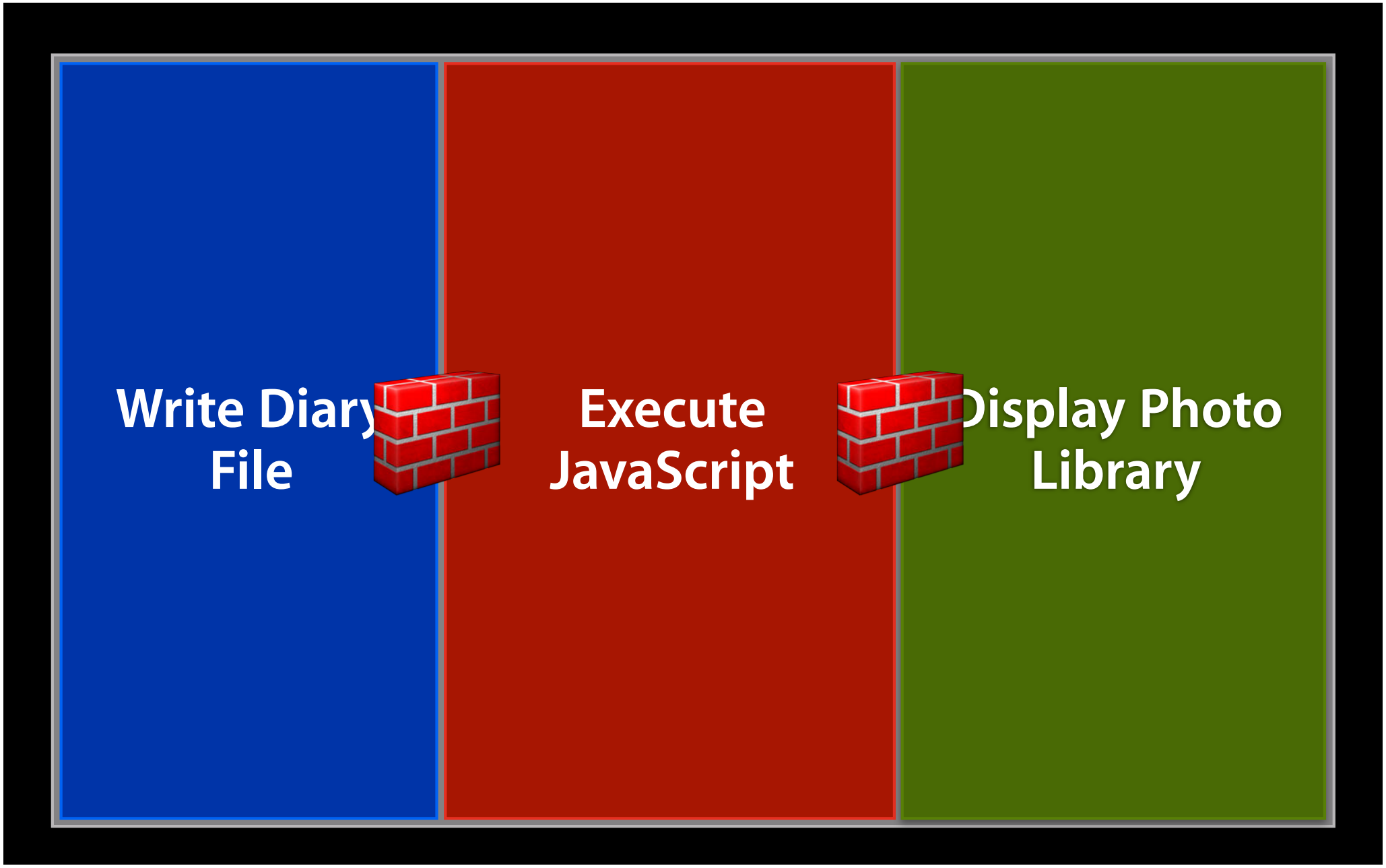

root
Me
Somebody Else


Memory


Write Diary File
Memory


Write Diary File
Memory
Display Photo Library


Write Diary File
Execute JavaScript
Display Photo Library


Write Diary File
Execute JavaScript
Display Photo Library

Mach messages

Sockets

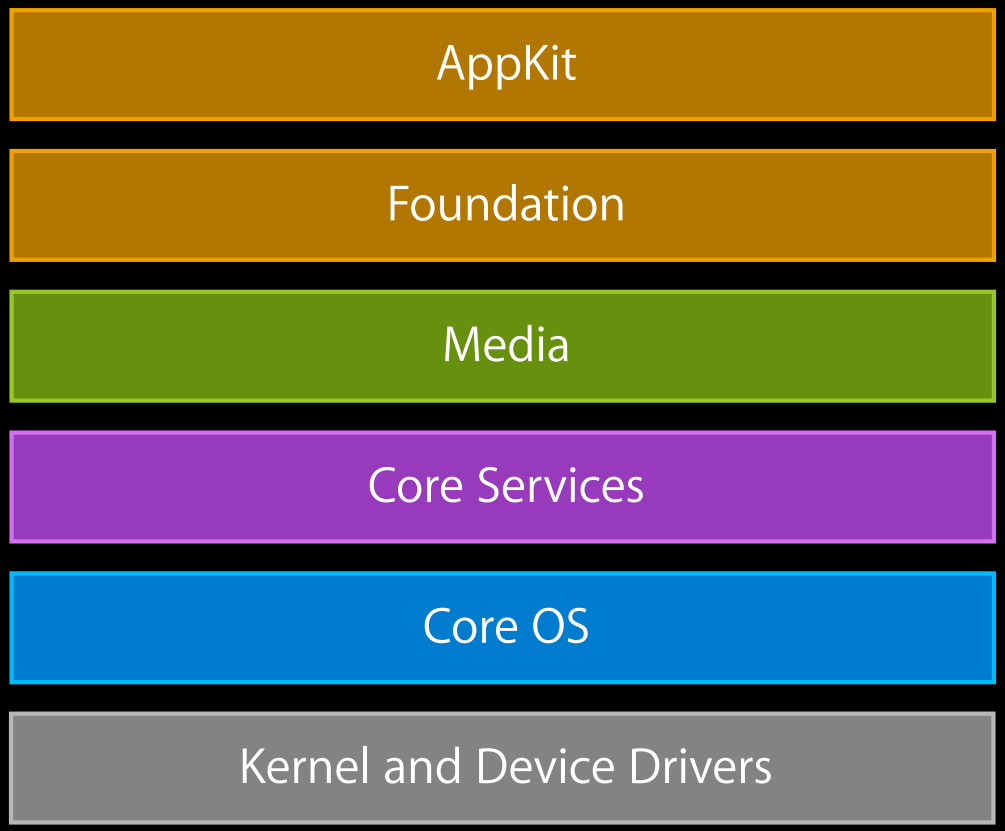
Distributed Objects

XPC

XPC

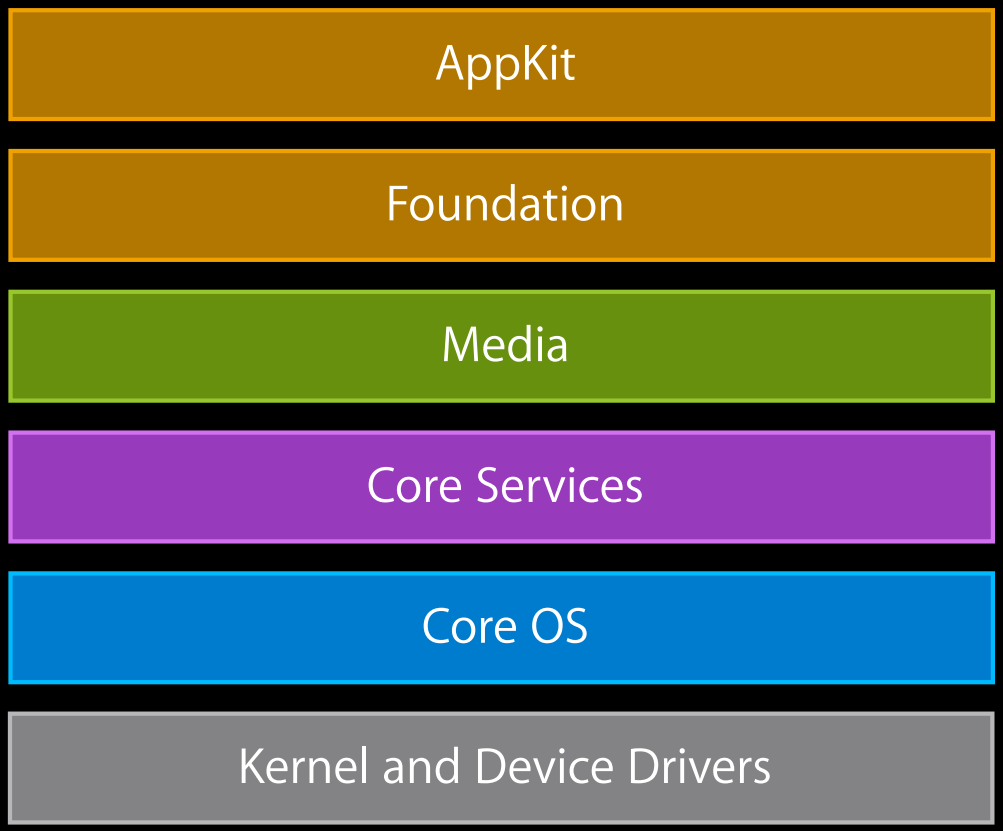
- Introduced in OS X 10.7 Lion
- Simplified the low level details of IPC
- C API with custom object code, containers, data types

Layers



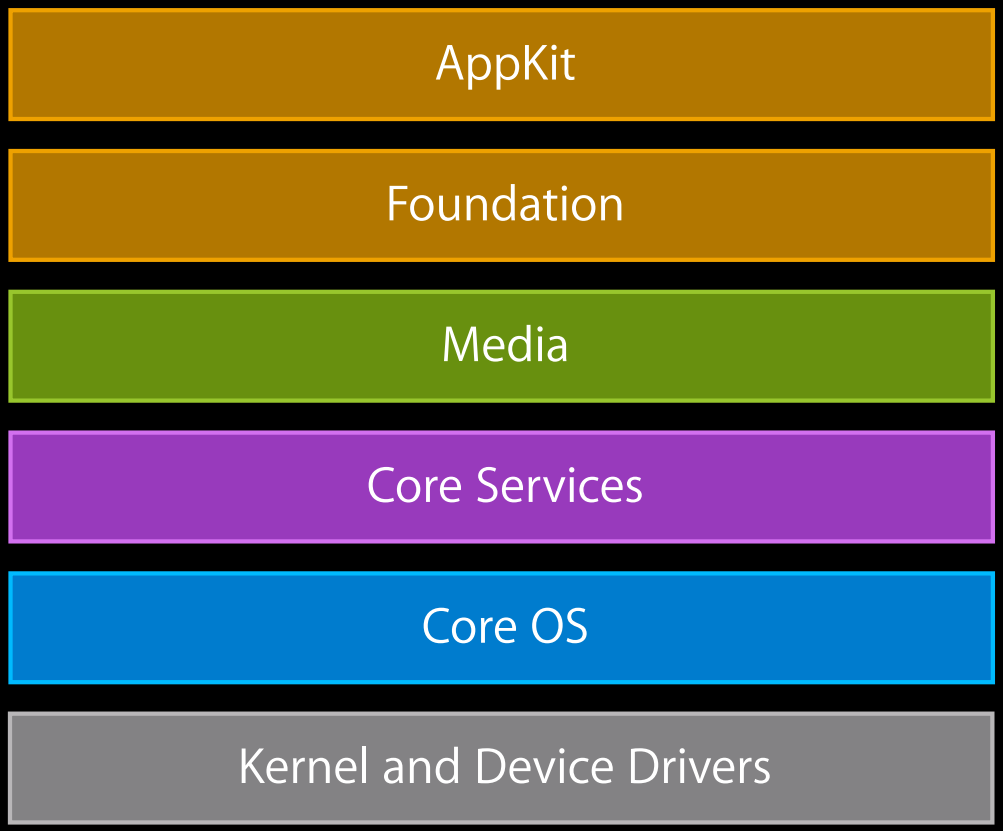
Adapted from OS X Technology Overview, Figure I-1

Layers



Adapted from OS X Technology Overview, Figure I-1

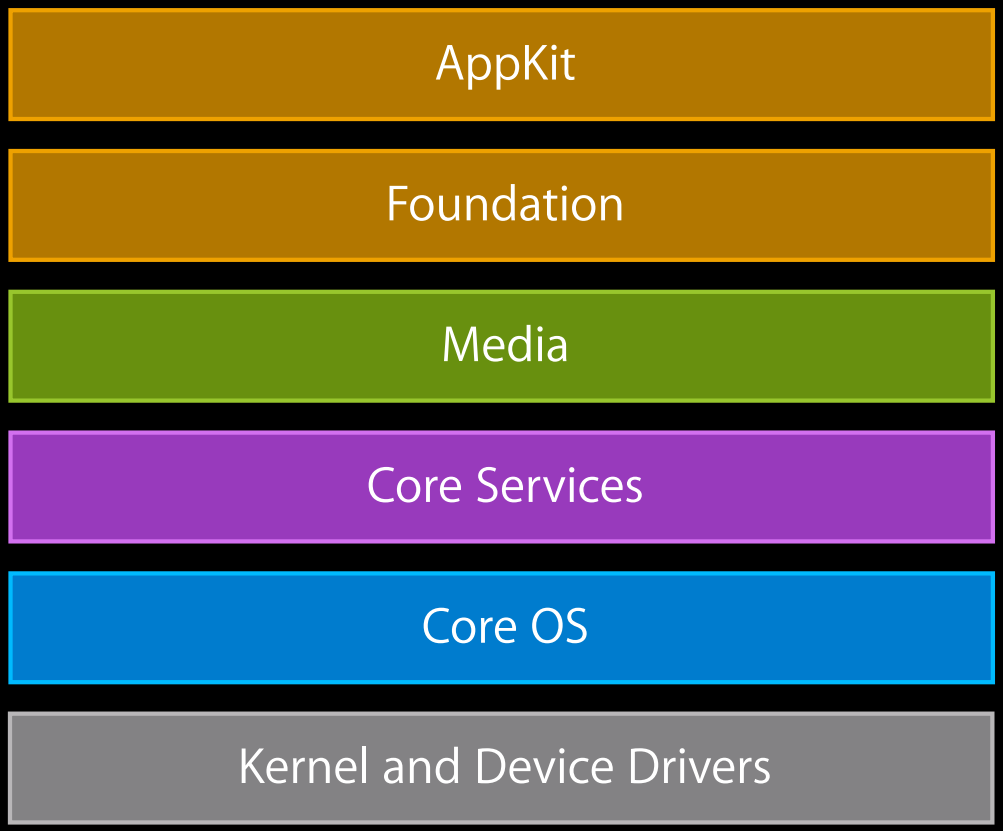
Layers



XPC

Adapted from OS X Technology Overview, Figure I-1

Layers



NSXPCConnection

XPC

Adapted from OS X Technology Overview, Figure I-1

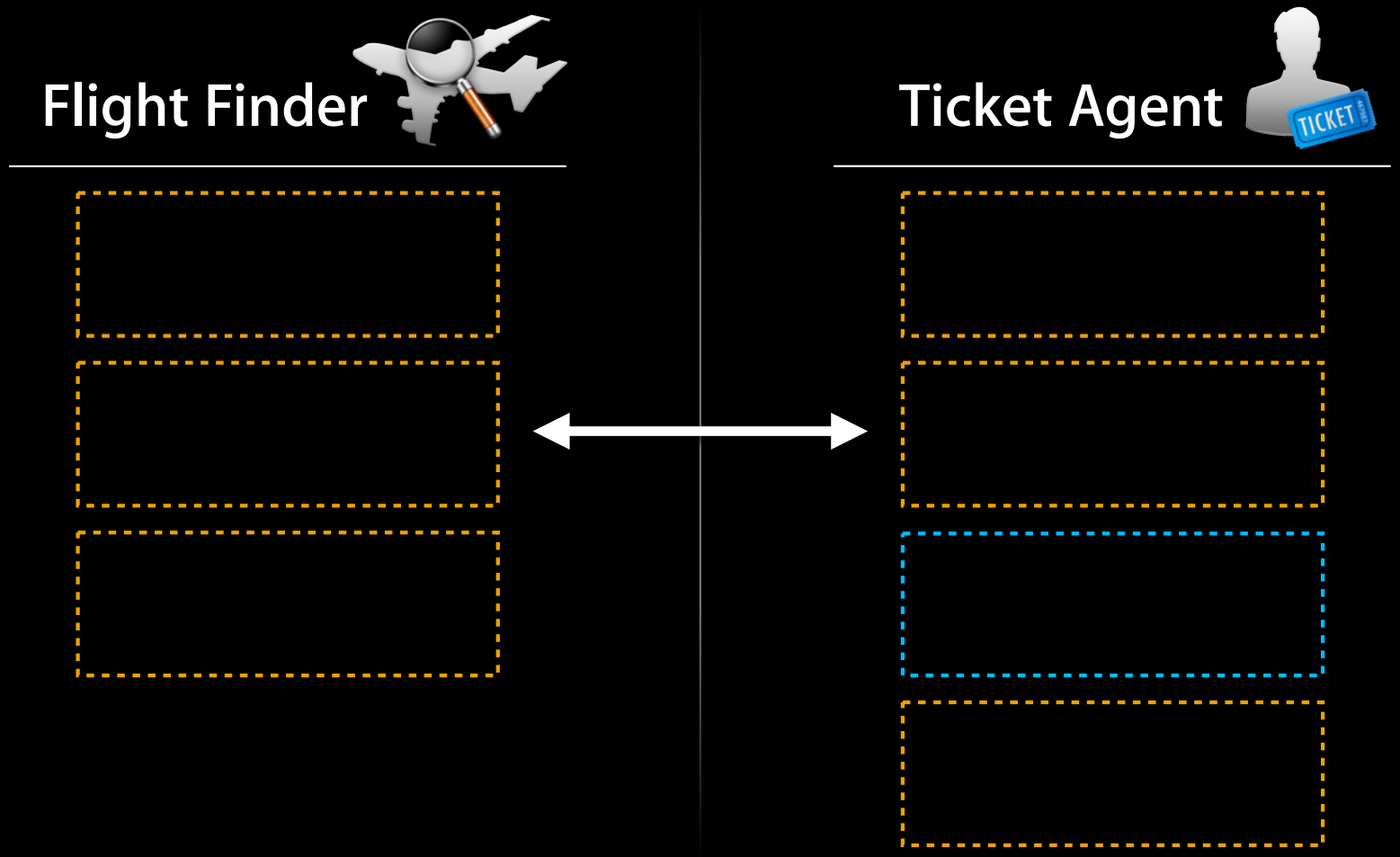
NSXPCConnection Design Goals

- Cocoa
- Simple
- Secure
- Asynchronous
- Modern

Demo

NSXPCConnection

Preview



What We Need

Interfaces

- Abstract away implementation details
- Provide a clear division of responsibilities
- Described by `@protocol`

Interfaces

Flight Finder

Get search input

Talk to agent

Show found tickets

Access calendar data

Ticket Agent

Find flights

Buy tickets

Interfaces

```
@protocol Agent



@end
```

Interfaces

```
@protocol Agent
- (void)checkIn;



@end
```

Interfaces

```
@protocol Agent
- (void)checkIn;
- (void)buyTicket:(NSString *)destination
           onDate:(NSDate *)date
          maxCost:(NSInteger)maxCost;

@end
```

Interfaces

```
@protocol Agent
- (void)checkIn;
- (void)buyTicket:(NSString *)destination
           onDate:(NSDate *)date
          maxCost:(NSInteger)maxCost;

@end

@protocol FlightFinder
- (void)setTicket:(Ticket *)ticket;
@end
```

Interfaces

```
@protocol Agent
- (void)checkIn;
- (void)buyTicket:(NSString *)destination
           onDate:(NSDate *)date
          maxCost:(NSInteger)maxCost
            reply:(void (^)(Ticket *))reply;
@end
```

Interfaces

```
@protocol Agent
- (void)checkIn;
- (void)buyTicket:(NSString *)destination
           onDate:(NSDate *)date
          maxCost:(NSInteger)maxCost
            reply:(void (^)(Ticket *))reply;
@end

NSXPCInterface *interface =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];
```

Protocol Methods

- Methods must return `void`
- Method arguments are of types:
  - Arithmetic types (e.g., `int`, `float`, `char`, `long long`, `NSInteger`)
  - `BOOL` type
  - C strings (`char *`)
  - C arrays containing above types
  - Structures containing above types
  - `NSSecureCoding`-compliant object

Interfaces

Flight Finder

Ticket Agent

Interfaces

Flight Finder

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

Interfaces

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

What We Need

What We Need

Connections

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

Connections

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

Connections

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Connections

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

Connections

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

Exported Object

```
@protocol Agent
- (void)checkIn;
- (void)buyTicket:(NSString *)destination
           onDate:(NSDate *)date
          maxCost:(NSInteger)maxCost
            reply:(void (^)(Ticket *))reply;
@end
```

Exported Object

```
@protocol Agent
- (void)checkIn;
- (void)buyTicket:(NSString *)destination
           onDate:(NSDate *)date
          maxCost:(NSInteger)maxCost
            reply:(void (^)(Ticket *))reply;
@end

@interface TicketAgent : NSObject <Agent>
@end
```

```
@implementation TicketAgent









@end
```

```
@implementation TicketAgent

- (void)checkIn {
  // do stuff
}









@end
```

```
@implementation TicketAgent

- (void)checkIn {
  // do stuff
}

- (void)buyTicket:(NSString *)destination
           onDate:(NSDate *)date
          maxCost:(NSInteger)maxCost
            reply:(void (^)(Ticket *))reply {


}

@end
```

```
@implementation TicketAgent

- (void)checkIn {
  // do stuff
}

- (void)buyTicket:(NSString *)destination
           onDate:(NSDate *)date
          maxCost:(NSInteger)maxCost
            reply:(void (^)(Ticket *))reply {
  Ticket *ticket = [Ticket ticketWith...];
  reply(ticket);
}

@end
```

Connections

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

Connections

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

Connections in the Application

```
NSXPCConnection *connection =
  [[NSXPCConnection alloc] initWithServiceName:@"com.apple.TicketAgent"];

connection.remoteObjectInterface =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

[connection resume];
```

Connections

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

Connections

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

Connections in the Service

```
NSXPCConnection *connection = ...;

connection.exportedInterface =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

TicketAgent *myTicketAgent = [TicketAgent new];
connection.exportedObject = myTicketAgent;

[connection resume];
```

Connections

- May have both exported object and remote object interface
- Lifetime of service is managed by XPC
- Invalidate connection in application when done to clean up

Connection Error Handling

- Interruption handler
  - Remote side has crashed or closed connection
  - NSXPCConnection instance still valid
  - May need to restore state

Connection Error Handling

- Interruption handler
  - Remote side has crashed or closed connection
  - NSXPCConnection instance still valid
  - May need to restore state
- Invalidation handler
  - Remote side has crashed or invalidate was called
  - NSXPCConnection instance no longer valid

What We Need

What We Need

Listeners

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

Listeners

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

NSXPCListener

Listeners

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

NSXPCListener

Creating a Listener

```
NSXPCListener *listener = [NSXPCListener serviceListener];

id <NSXPCListenerDelegate> delegate = [MyDelegate new];
listener.delegate = delegate;

[listener resume];
```

Implementing the Delegate

```
- (BOOL)listener:(NSXPCListener *)listener
  shouldAcceptNewConnection:(NSXPCConnection *)connection {









}
```

Implementing the Delegate

```
- (BOOL)listener:(NSXPCListener *)listener
  shouldAcceptNewConnection:(NSXPCConnection *)connection {

  // Configure the connection
  connection.exportedInterface =
    [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];
  connection.exportedObject = [TicketAgent new];

  // Resume or the connection will not receive messages
  [connection resume];

  return YES;
}
```

LaunchAgents and LaunchDaemons

- NSXPCListener

```
[[NSXPCListener alloc] initWithMachServiceName:@"..."];
```

- NSXPCConnection

```
[[NSXPCConnection alloc] initWithMachServiceName:@"..." options:0];
```

- Privileged daemons must use `NSXPCConnectionPrivileged` option
- Mach service listeners do not take control in `resume` method

What We Need

What We Need

Messages

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

NSXPCListener

Messages

Flight Finder

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Remote Object Proxy
id <Agent>

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

NSXPCListener

Messages

Flight Finder

NSXPCInterface
@protocol(Agent)
"Remote Object Interface"

NSXPCConnection

Remote Object Proxy
id <Agent>

Ticket Agent

NSXPCInterface
@protocol(Agent)
"Exported Interface"

NSXPCConnection

Exported Object
id <Agent>

NSXPCListener

Simple Messages

```
NSXPCConnection *c = ...;

[[c remoteObjectProxy] checkIn];
```

Simple Messages

```
NSXPCConnection *c = ...;

[[c remoteObjectProxy] checkIn];
```

Messages with Replies

```
NSXPCConnection *c = ...;

[[c remoteObjectProxy] buyTicket:@"NYC"
                        onDate:nextTues
                       maxCost:500
                         reply:^(Ticket *tk) {
  NSLog(@"Got ticket: %@", tk);
}];
```

Messages with Replies and Error Handlers

```
NSXPCConnection *c = ...;

[[c remoteObjectProxyWithErrorHandler:^(NSError *err) {

  // The other side probably crashed
  NSLog(@"Oops!");

}] buyTicket:@"NYC" onDate:nextTues maxCost:500 reply:^(Ticket *tk) {

  NSLog(@"Got ticket: %@", tk);

}];
```

Messages with Replies and Error Handlers

Exactly one of these blocks will be called

```
NSXPCConnection *c = ...;

[[c remoteObjectProxyWithErrorHandler:^(NSError *err) {

    // The other side probably crashed
    NSLog(@"Oops!");

}] buyTicket:@"NYC" onDate:nextTues maxCost:500 reply:^(Ticket *tk) {

    NSLog(@"Got ticket: %@", tk);

}];
```

Messages

Flight Finder

Remote Object Proxy
id <Agent>

Ticket Agent

Exported Object
id <Agent>

Messages

Flight Finder

`-buyTicket:...`

Remote Object Proxy
id <Agent>

Ticket Agent

Exported Object
id <Agent>

Messages

Flight Finder

Ticket Agent

–buyTicket:...

Remote Object Proxy
id <Agent>

Exported Object
id <Agent>

Reply Block

Messages

Flight Finder

Ticket Agent

–buyTicket:...

Remote Object Proxy
id <Agent>

Exported Object
id <Agent>

Reply Block

Messages

Flight Finder

Ticket Agent

–buyTicket:...

Remote Object Proxy
id <Agent>

Exported Object
id <Agent>

Reply Block

Reply Block Proxy

Messages

Flight Finder

Ticket Agent

–buyTicket:...

Remote Object Proxy
id <Agent>

Exported Object
id <Agent>

Reply Block

Reply Block Proxy

reply(ticket);

Messages

Flight Finder

Ticket Agent

`-buyTicket:...`

Remote Object Proxy
id <Agent>

Exported Object
id <Agent>

Reply Block

Reply Block Proxy

`reply(ticket);`

Messages

- Proxies
  - Lightweight
  - Immutable
  - Created from NSXPCConnection or another proxy
    - `- (id)remoteObjectProxy;`
    - `- (id)remoteObjectProxyWithErrorHandler:(void (^)(NSError *))handler;`

Messages

- Proxies
  - Lightweight
  - Immutable
  - Created from NSXPCConnection or another proxy
    - `- (id)remoteObjectProxy;`
    - `- (id)remoteObjectProxyWithErrorHandler:(void (^)(NSError *))handler;`
- Messages delivered on private serial queue

What We Need

What We Need

Demo

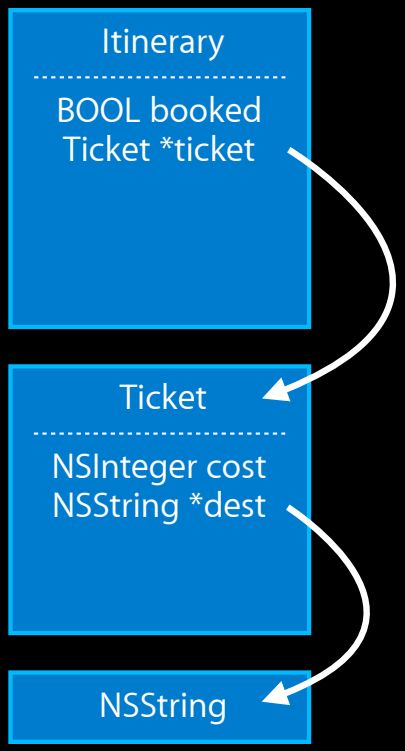
Object Coding

NSCoding Review

- Two parts
  - `NSCoder` subclass
  - `NSCoding`-conforming class
- Two activities
  - Encoding
  - Decoding

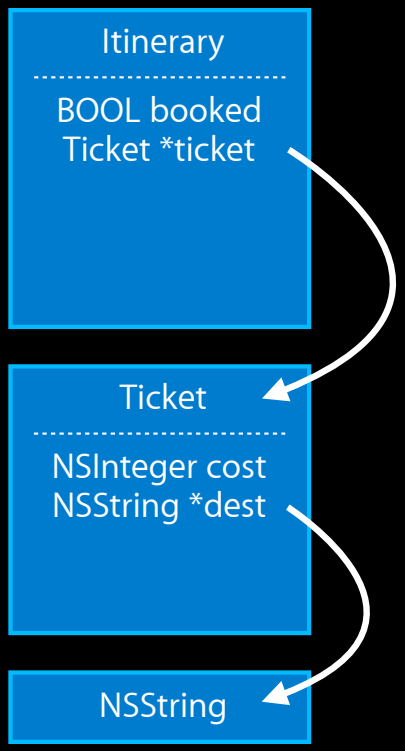
NSCoding Review

Encoding



NSCoding Review

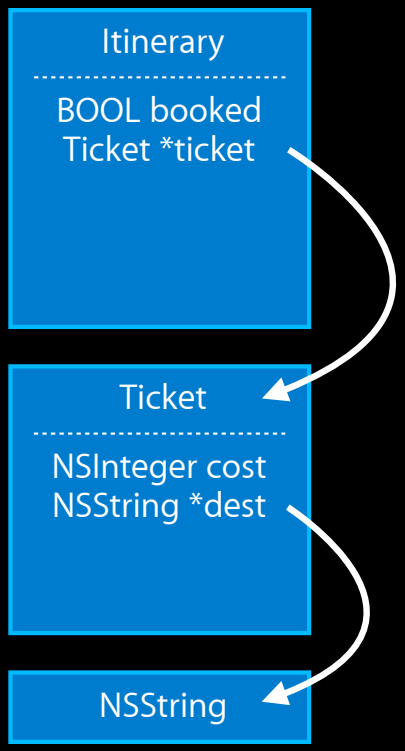
Encoding




```
- (void)encodeWithCoder:(NSCoder *)coder {
    [coder encodeBool:booked forKey:@"booked"];
    [coder encodeObject:ticket forKey:@"ticket"];
}
```

NSCoding Review

Encoding

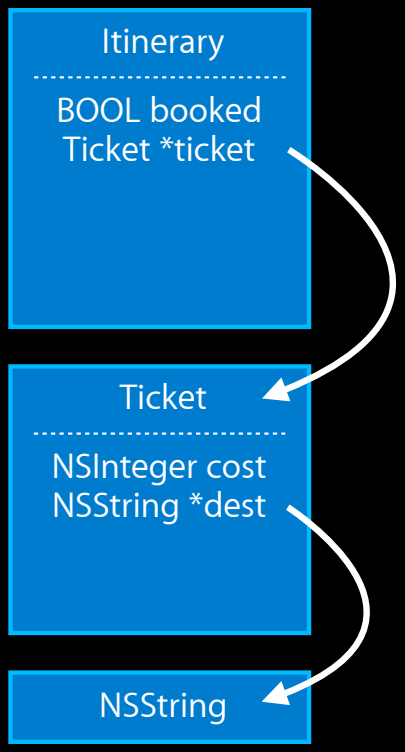



```
- (void)encodeWithCoder:(NSCoder *)coder {
  [coder encodeBool:booked forKey:@"booked"];
  [coder encodeObject:ticket forKey:@"ticket"];
}
```

```
- (void)encodeWithCoder:(NSCoder *)coder {
  [coder encodeInteger:cost forKey:@"cost"];
  [coder encodeObject:dest forKey:@"dest"];
}
```

NSCoding Review

Encoding




```
- (void)encodeWithCoder:(NSCoder *)coder {
  [coder encodeBool:booked forKey:@"booked"];
  [coder encodeObject:ticket forKey:@"ticket"];
}
```

```
- (void)encodeWithCoder:(NSCoder *)coder {
  [coder encodeInteger:cost forKey:@"cost"];
  [coder encodeObject:dest forKey:@"dest"];
}
```

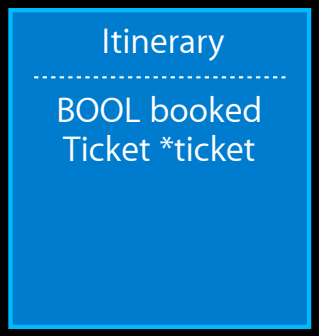
```
- (void)encodeWithCoder:(NSCoder *)coder;
```

NSCoding Review

Decoding

NSCoding Review

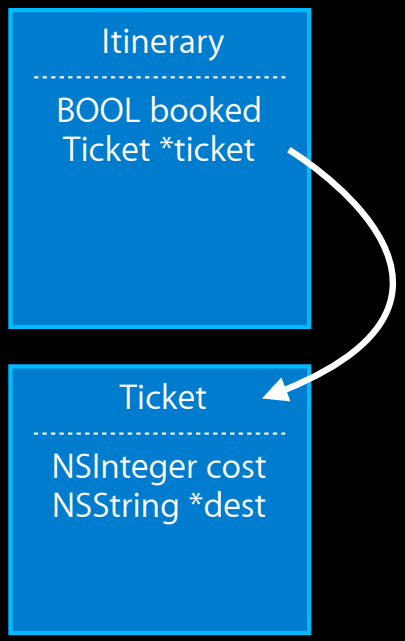
Decoding




```
- (id)initWithCoder:(NSCoder *)coder {
  self = [super init];
  booked = [coder decodeBoolForKey:@"booked"];
  ticket = [coder decodeObjectForKey:@"ticket"];
  return self;
}
```

NSCoding Review

Decoding

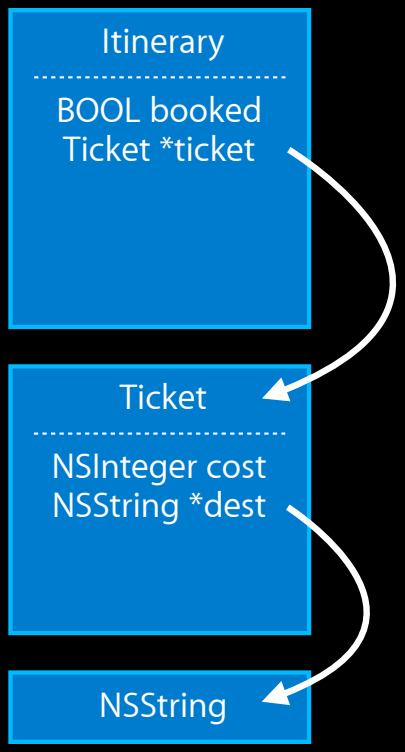



```
- (id)initWithCoder:(NSCoder *)coder {
  self = [super init];
  booked = [coder decodeBoolForKey:@"booked"];
  ticket = [coder decodeObjectForKey:@"ticket"];
  return self;
}
```

```
- (id)initWithCoder:(NSCoder *)coder {
  self = [super init];
  cost = [coder decodeIntegerForKey:@"cost"];
  dest = [coder decodeObjectForKey:@"dest"];
  return self;
}
```

NSCoding Review

Decoding




```
- (id)initWithCoder:(NSCoder *)coder {
  self = [super init];
  booked = [coder decodeBoolForKey:@"booked"];
  ticket = [coder decodeObjectForKey:@"ticket"];
  return self;
}
```

```
- (id)initWithCoder:(NSCoder *)coder {
  self = [super init];
  cost = [coder decodeIntegerForKey:@"cost"];
  dest = [coder decodeObjectForKey:@"dest"];
  return self;
}
```

```
- (id)initWithCoder:(NSCoder *)coder;
```

NSXPCConnection Coding

How objects are copied between processes

- Uses the same `NSCoding` design pattern
- On message send, all arguments are encoded
- On message receive, all arguments are decoded
- `NSXPCConnection` uses a new `NSCoder` subclass

Argument Encoding

- When a message is sent:

```
[[c remoteObjectProxy] buyTicket:@"NYC"
                          onDate:nextTues
                         maxCost:500
                           reply:^(Ticket *tk) {
    // reply handling
}];
```

Argument Encoding

- When a message is sent:

```
[[c remoteObjectProxy] buyTicket:@"NYC"
                          onDate:nextTues
                         maxCost:500
                           reply:^(Ticket *tk) {
    // reply handling
}];
```

- The arguments are encoded:

```
[coder encodeObject:@"NYC" forKey:@"arg1"];
[coder encodeObject:nextTues forKey:@"arg2"];
[coder encodeInteger:500 forKey:@"arg3"];
```

Argument Encoding

- When a message is sent:

```
[[c remoteObjectProxy] buyTicket:@"NYC"
                          onDate:nextTues
                         maxCost:500
                           reply:^(Ticket *tk) {
    // reply handling
}];
```

- The arguments are encoded:

```
[coder encodeObject:@"NYC" forKey:@"arg1"];
[coder encodeObject:nextTues forKey:@"arg2"];
[coder encodeInteger:500 forKey:@"arg3"];
```

- The reply block is held onto until the reply comes back

Argument Decoding

- On receiving a message, arguments are decoded:

```
id arg1 = [decoder decodeObjectForKey:@"arg1"];
id arg2 = [decoder decodeObjectForKey:@"arg2"];
NSInteger arg3 = [decoder decodeIntegerForKey:@"arg3"];
```

Argument Decoding

- On receiving a message, arguments are decoded:

```
id arg1 = [decoder decodeObjectForKey:@"arg1"];
id arg2 = [decoder decodeObjectForKey:@"arg2"];
NSInteger arg3 = [decoder decodeIntegerForKey:@"arg3"];
```

- The reply block proxy is created:

```
id replyBlock = [[ReplyBlockProxy alloc] init];
```

Argument Decoding

- On receiving a message, arguments are decoded:

```
id arg1 = [decoder decodeObjectForKey:@"arg1"];
id arg2 = [decoder decodeObjectForKey:@"arg2"];
NSInteger arg3 = [decoder decodeIntegerForKey:@"arg3"];
```

- The reply block proxy is created:

```
id replyBlock = [[ReplyBlockProxy alloc] init];
```

- The message is sent to the exported object

```
[exportedObject buyTicket:arg1
                   onDate:arg2
                  maxCost:arg3
                    reply:replyBlock];
```

Argument Decoding

- On receiving a message, arguments are decoded:

```
id arg1 = [decoder decodeObjectForKey:@"arg1"];
```

Argument Decoding

- On receiving a message, arguments are decoded:

```
id arg1 = [decoder decodeObjectForKey:@"arg1"];
```

- What kind of object is this?
  - Class comes from archive
  - Archive comes from remote process
  - Remote process should be untrusted

Argument Decoding

- On receiving a message, arguments are decoded:

```
id arg1 = [decoder decodeObjectForKey:@"arg1"];
```

- What kind of object is this?
  - Class comes from archive
  - Archive comes from remote process
  - Remote process should be untrusted
- Need a way to specify expected class

Class Information From Interface

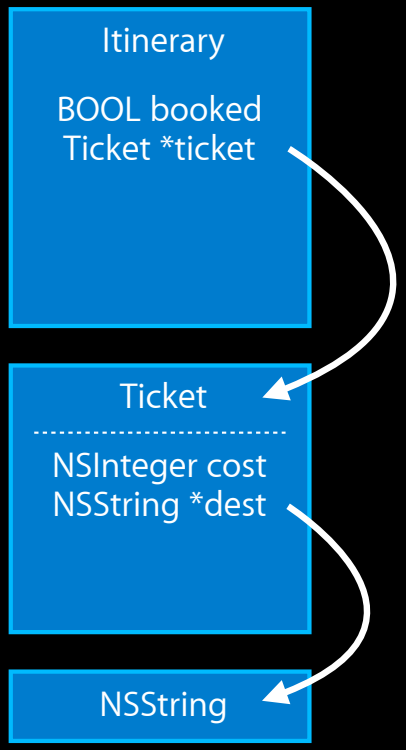
- Protocol used for `NSXPCInterface`

```
@protocol Agent
- (void)checkIn;
- (void)buyTicket:(NSString *)destination
           onDate:(NSDate *)date
          maxCost:(NSInteger)maxCost
            reply:(void (^)(Ticket *))reply;
@end
```

- Metadata provided by Mountain Lion clang compiler

Additional Classes

- Objects usually come in graphs
- `NSSecureCoding` protocol and new `NSCoder` methods provide info beyond first object



Checking Decoded Classes

```
- (id)initWithCoder:(NSCoder *)coder {
  self = [super init];
  Ticket *result = [decoder decodeObjectForKey:@"ticketKey"];

  if (![result isKindOfClass:[Ticket class]]) {
    panic(); // Too late! result is already decoded
  }
  return self;
}
```

Checking Decoded Classes

```
- (id)initWithCoder:(NSCoder *)coder {
  self = [super init];
  Ticket *result = [decoder decodeObjectForKey:@"ticketKey"];

  if (![result isKindOfClass:[Ticket class]]) {
    panic(); // Too late! result is already decoded
  }
  return self;
}
```

Checking Decoded Classes

New NSCoder methods

Checking Decoded Classes

New NSCoder methods

- Single class
  - `- (id)decodeObjectOfClass:(Class)aClass forKey:(NSString *)key;`

Checking Decoded Classes

New NSCoder methods

- Single class
  - `- (id)decodeObjectOfClass:(Class)aClass forKey:(NSString *)key;`
- Property list type
  - `- (id)decodePropertyListForKey:(NSString *)key;`

Checking Decoded Classes

New NSCoder methods

- Single class
  - `- (id)decodeObjectOfClass:(Class)aClass forKey:(NSString *)key;`
- Property list type
  - `- (id)decodePropertyListForKey:(NSString *)key;`
- One of several classes or collection
  - `- (id)decodeObjectOfClasses:(NSSet *)classes forKey:(NSString *)key;`

Checking Decoded Classes

New NSCoder methods

- Single class
  - `- (id)decodeObjectOfClass:(Class)aClass forKey:(NSString *)key;`
- Property list type
  - `- (id)decodePropertyListForKey:(NSString *)key;`
- One of several classes or collection
  - `- (id)decodeObjectOfClasses:(NSSet *)classes forKey:(NSString *)key;`

```
NSSet *classes =
  [NSSet setWithObjects:[Ticket class], [NSArray class], nil];
NSArray *result =
  [decoder decodeObjectOfClasses:classes forKey:@"ticketListKey"];
```

Adopting Secure Coding

```
@interface Ticket : NSObject <NSSecureCoding>
@property (copy) NSString *destination;
@end
```

```
@implementation Ticket









@end
```

```objc
@implementation Ticket

- (void)encodeWithCoder:(NSCoder *)coder {
  [coder encodeObject:_destination forKey:@"dest"];
}









@end
```

```
@implementation Ticket

- (void)encodeWithCoder:(NSCoder *)coder {
  [coder encodeObject:_destination forKey:@"dest"];
}

- (id)initWithCoder:(NSCoder *)coder {



}


@end
```

```
@implementation Ticket

- (void)encodeWithCoder:(NSCoder *)coder {
  [coder encodeObject:_destination forKey:@"dest"];
}

- (id)initWithCoder:(NSCoder *)coder {
  self = [super init];
  _destination = [coder decodeObjectOfClass:[NSString class]
                                     forKey:@"dest"];
  return self;
}



@end
```

```objc
@implementation Ticket

- (void)encodeWithCoder:(NSCoder *)coder {
  [coder encodeObject:_destination forKey:@"dest"];
}

- (id)initWithCoder:(NSCoder *)coder {
  self = [super init];
  _destination = [coder decodeObjectOfClass:[NSString class]
                                     forKey:@"dest"];
  return self;
}

+ (BOOL)supportsSecureCoding { return YES; }

@end
```

Adopting Secure Coding

Adopting Secure Coding

- `NSSecureCoding` protects against specific security issue

Adopting Secure Coding

- `NSSecureCoding` protects against specific security issue
- Other vulnerabilities are still possible
  - Buffer overrun
  - False trust in remote process

Adopting Secure Coding

- `NSSecureCoding` protects against specific security issue
- Other vulnerabilities are still possible
  - Buffer overrun
  - False trust in remote process
- Review code in `-initWithCoder:`

Adopting Secure Coding

- `NSSecureCoding` protects against specific security issue
- Other vulnerabilities are still possible
  - Buffer overrun
  - False trust in remote process
- Review code in `-initWithCoder:`
- If you override `-initWithCoder:` you must also override `+supportsSecureCoding`

Top Level Collections

Top Level Collections

- Collections do not know any details about content

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;
@end
```

Top Level Collections

- Collections do not know any details about content

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;
@end
```

- Property list types are automatically whitelisted
  - NSArray, NSDictionary
  - NSString, NSData, NSDate, NSNumber

Top Level Collections

- Collections do not know any details about content

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;
@end
```

- Property list types are automatically whitelisted
  - NSArray, NSDictionary
  - NSString, NSData, NSDate, NSNumber
- Other types must be set on the interface

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;

@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];
```

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;

@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

NSSet *expected = [NSSet setWithObject:[Ticket class]];
```

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;

@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

NSSet *expected = [NSSet setWithObject:[Ticket class]];

[ifc setClasses:
    forSelector:
  argumentIndex:
        ofReply:  ];
```

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;

@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

NSSet *expected = [NSSet setWithObject:[Ticket class]];

[ifc setClasses:expected
    forSelector:
  argumentIndex:
        ofReply:  ];
```

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;

@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

NSSet *expected = [NSSet setWithObject:[Ticket class]];

[ifc setClasses:expected
    forSelector:@selector(checkInFamily:)
  argumentIndex:
        ofReply:  ];
```

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;

@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

NSSet *expected = [NSSet setWithObject:[Ticket class]];

[ifc setClasses:expected
    forSelector:@selector(checkInFamily:)
  argumentIndex:0
        ofReply:  ];
```

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;

@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

NSSet *expected = [NSSet setWithObject:[Ticket class]];

[ifc setClasses:expected
    forSelector:@selector(checkInFamily:)
  argumentIndex:0
        ofReply:NO];
```

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;
- (void)getLotsOfTickets:(void (^)(NSArray *))reply;
@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

NSSet *expected = [NSSet setWithObject:[Ticket class]];

[ifc setClasses:expected
    forSelector:@selector(getLotsOfTickets:)
  argumentIndex:
        ofReply:   ];
```

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;
- (void)getLotsOfTickets:(void (^)(NSArray *))reply;
@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

NSSet *expected = [NSSet setWithObject:[Ticket class]];

[ifc setClasses:expected
    forSelector:@selector(getLotsOfTickets:)
  argumentIndex:
        ofReply:YES];
```

Setting Up Additional Classes

```
@protocol Agent
- (void)checkInFamily:(NSArray *)tickets;
- (void)getLotsOfTickets:(void (^)(NSArray *))reply;
@end

NSXPCInterface *ifc =
  [NSXPCInterface interfaceWithProtocol:@protocol(Agent)];

NSSet *expected = [NSSet setWithObject:[Ticket class]];

[ifc setClasses:expected
    forSelector:@selector(getLotsOfTickets:)
  argumentIndex:0
        ofReply:YES];
```

Design Patterns

Service Patterns

Service Patterns

- Little state, mostly functional and short-lived
  - One singleton thread-safe object
  - Implements `NSXPCListenerDelegate` and exported object protocol

Service Patterns

- Little state, mostly functional and short-lived
  - One singleton thread-safe object
  - Implements `NSXPCListenerDelegate` and exported object protocol
- Lots of state, mostly long-lived
  - `NSXPCListenerDelegate` responsibility in one object
  - One new exported object per connection

Application Patterns

- Asynchronous UI updates
- Separation of interface and implementation

Demo

- Add XPC service to project
- Move code from main application to service
- Debug a service

Demo

Summary

- Multiprocess applications are here
- NSXPCConnection helps you connect them
  - Works with your own objects and interfaces
  - Secure and modern
- Three main components
  - Interfaces
  - Connections
  - Listeners

More Information

Mike Jurewitz
Developer Tools and Performance Evangelist
jurewitz@apple.com

Documentation
Daemons and Services Programming Guide
http://developer.apple.com/

XPC man Pages
man xpc

SandboxedFetch2 Example
http://developer.apple.com/library/wwdc/mac/samplecode/SandboxedFetch2/

Related Sessions

Asynchronous Design Patterns with Blocks, GCD, and XPC	Pacific Heights Friday 9:00AM

Labs

Cocoa and XPC Lab	Essentials Lab A Friday 10:15AM


WWDC2012