



ZOOM OUT

BREW 2008 CONFERENCE

BREW® Mobile Platform

Operating System Services Overview

Ramesh Chandrasekhar, Principal Engineer

Qualcomm Incorporated

QUALCOMM



BREW Mobile Platform OS Services

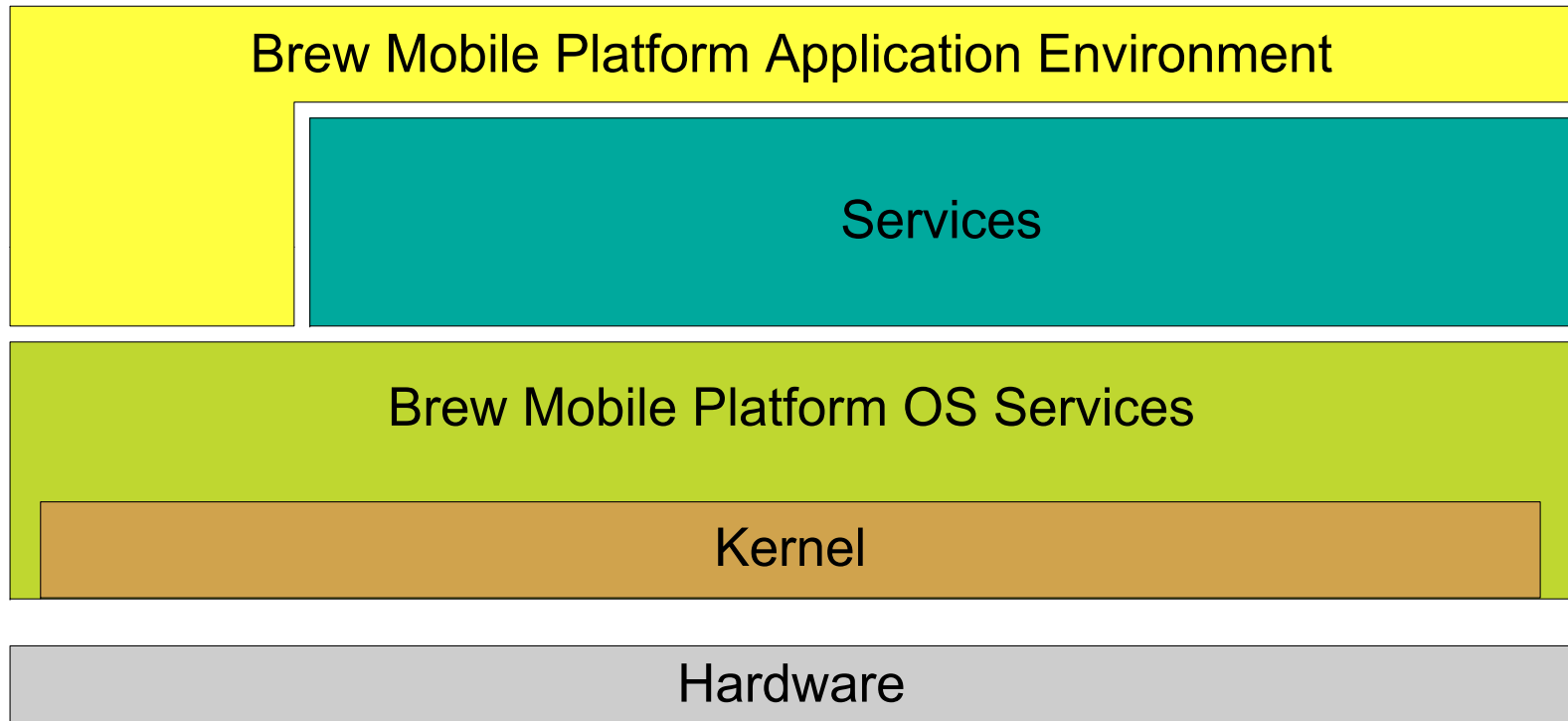
- Requirements
 - API for basic OS services
 - Kernel services
 - File system APIs
 - Support for loadable code
 - Component Infrastructure
 - Foundation for service components
 - Portability



BREW Mobile Platform OS Services

- Usage
 - Foundation for Brew Mobile Platform Application Environment
 - Enable Services and BREW to be:
 - Developed in a more modular and portable way
 - Utilize memory protection
 - Multi-processor communication

BREW Mobile Platform Architecture



Kernel Model: Processes and Threads

A running system consists of threads executing in a kernel and some number of user processes

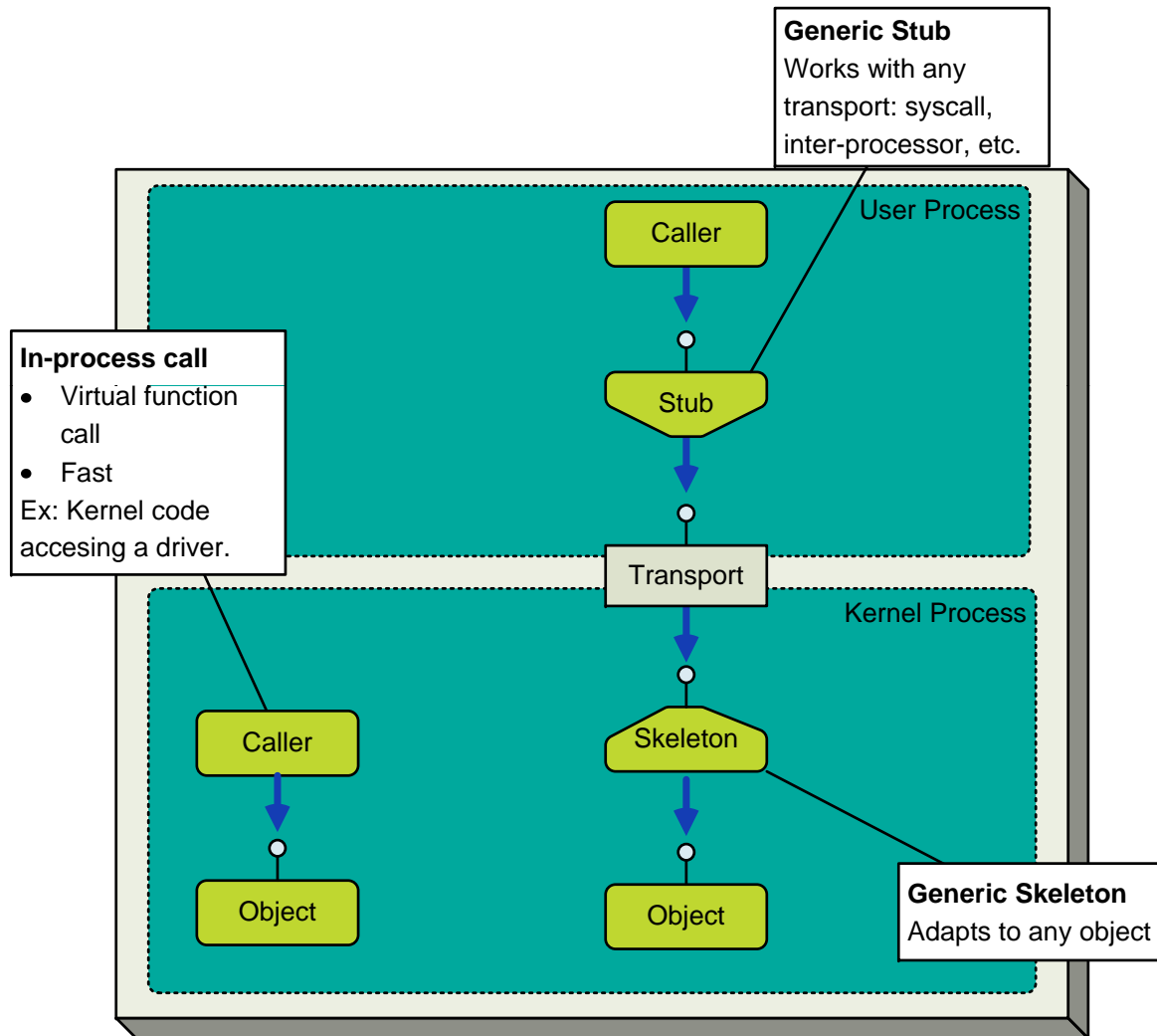
- User process = confined execution environment
 - Memory: Each process can read and/or write only the memory it has been granted
 - Kernel objects: Each process can control system resources (e.g. files, devices) only when it has been granted them
- Kernel process = privileged environment
 - The kernel controls user processes, enforces restrictions, and decides which interactions to allow between them
- Threads
 - Threads live in processes: code executing in a thread is limited in its access to memory and kernel services according to the process in which it lives



Kernel Model: Capability-based Security

- Everything is an object
 - Processes, threads, memory regions, files, etc.
- Mechanism, not policy
 - Kernel enforces that processes can only access objects they have been granted.
 - Kernel does not decide to whom objects will be granted.
 - Kernel does not dictate what objects will do. Fine-grained authority can be delegated.
- No “ambient authority”
 - Processes do not have inherent power or authority over the system.
 - They only have objects they have been granted, and all kernel requests operate on objects.

Remote Invocation





OS Services

- Environment
- Signals
- Threads
- Critical Sections
- Memory Management
- Servers, Programs (Processes)
- Timers
- Utilities
- Async message passing
- Interrupt Controller



Environment Object

- Every object is given an environment object when it is created
- An environment exposes IEnv
 - Access to Component Infrastructure :
IEnv_CreateInstance()
 - Use this to create objects implemented in other components
 - Kernel objects are obtained through same mechanism
 - Access to Heap services : IEnv_ErrMalloc(), etc.



Signals

- Clients (the notified, or observers) create signal objects to wait on.
- Clients deliver signal objects to servers (the notifiers, or observed).
- Upon creation, signals are automatically enabled.
- Server can notify the client of an event or state change.
- Signals are one-shot
- Signals are dataless (pure notifications)



Threads

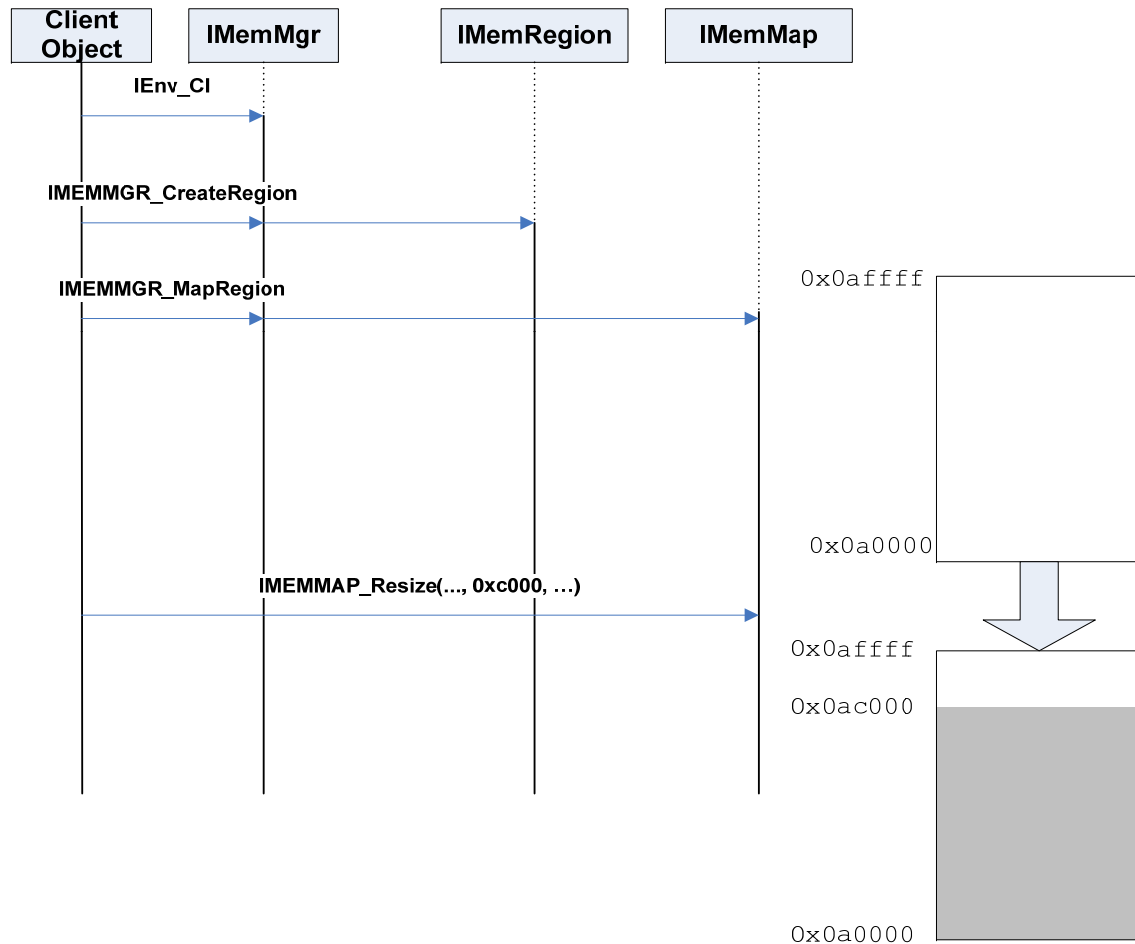
- Thread is an execution entity known to scheduler
- Priority based pre-emptive threading
- User allocated stack
- Key features supported
 - Start the thread and provide a start function
 - Provide a signal to the thread to be set when the thread exits
 - Kill a thread
 - Get the exit code for a thread that has exit



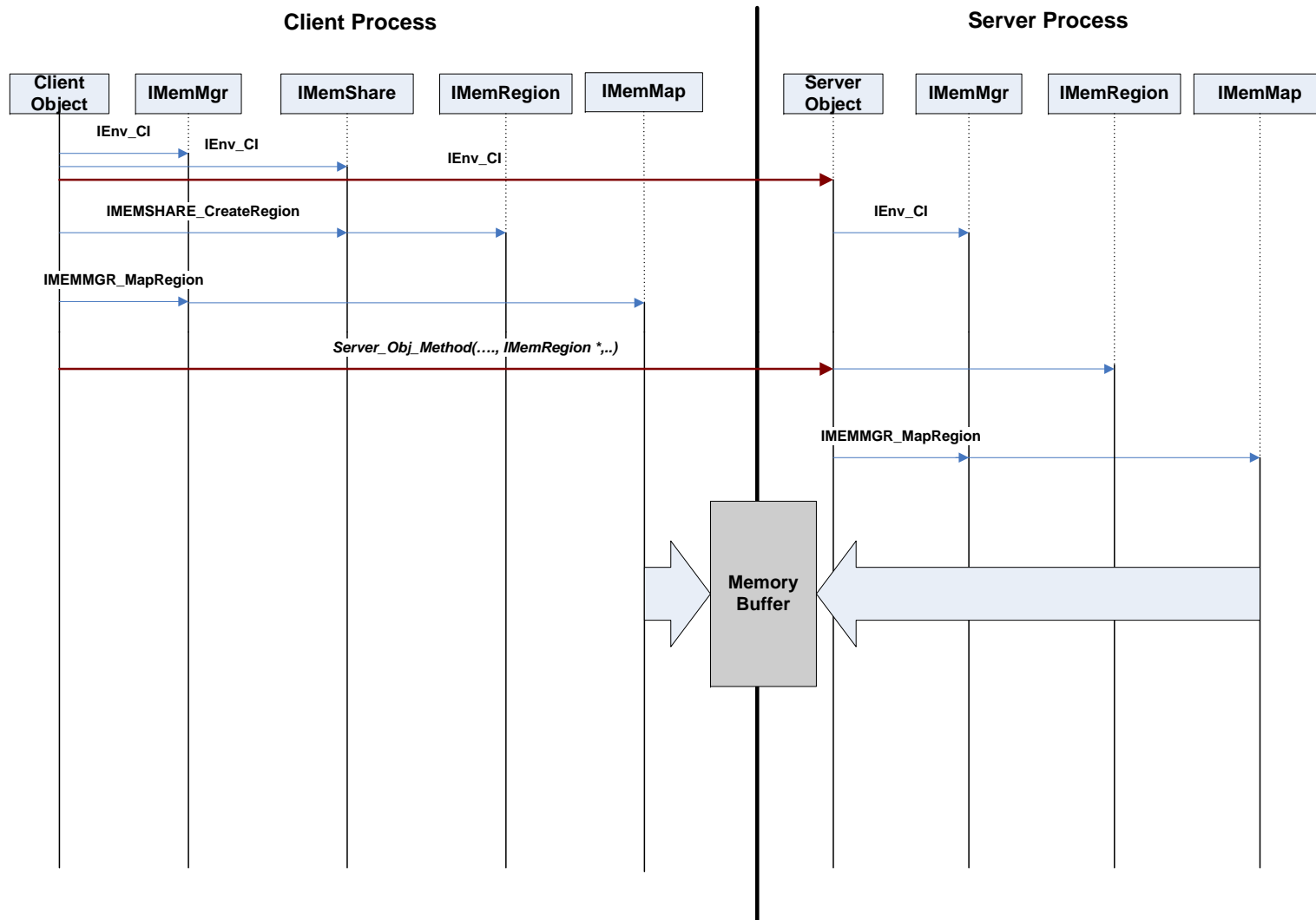
Critical Sections

- A single thread access to the code protected by Critical Section.
 - Performs priority elevation of the owner thread when higher priority thread is waiting.
 - When multiple threads are waiting, the one with highest priority wins the critical section when available.

In-process memory maps



Client-Server Shared Memory Scenario





Timers

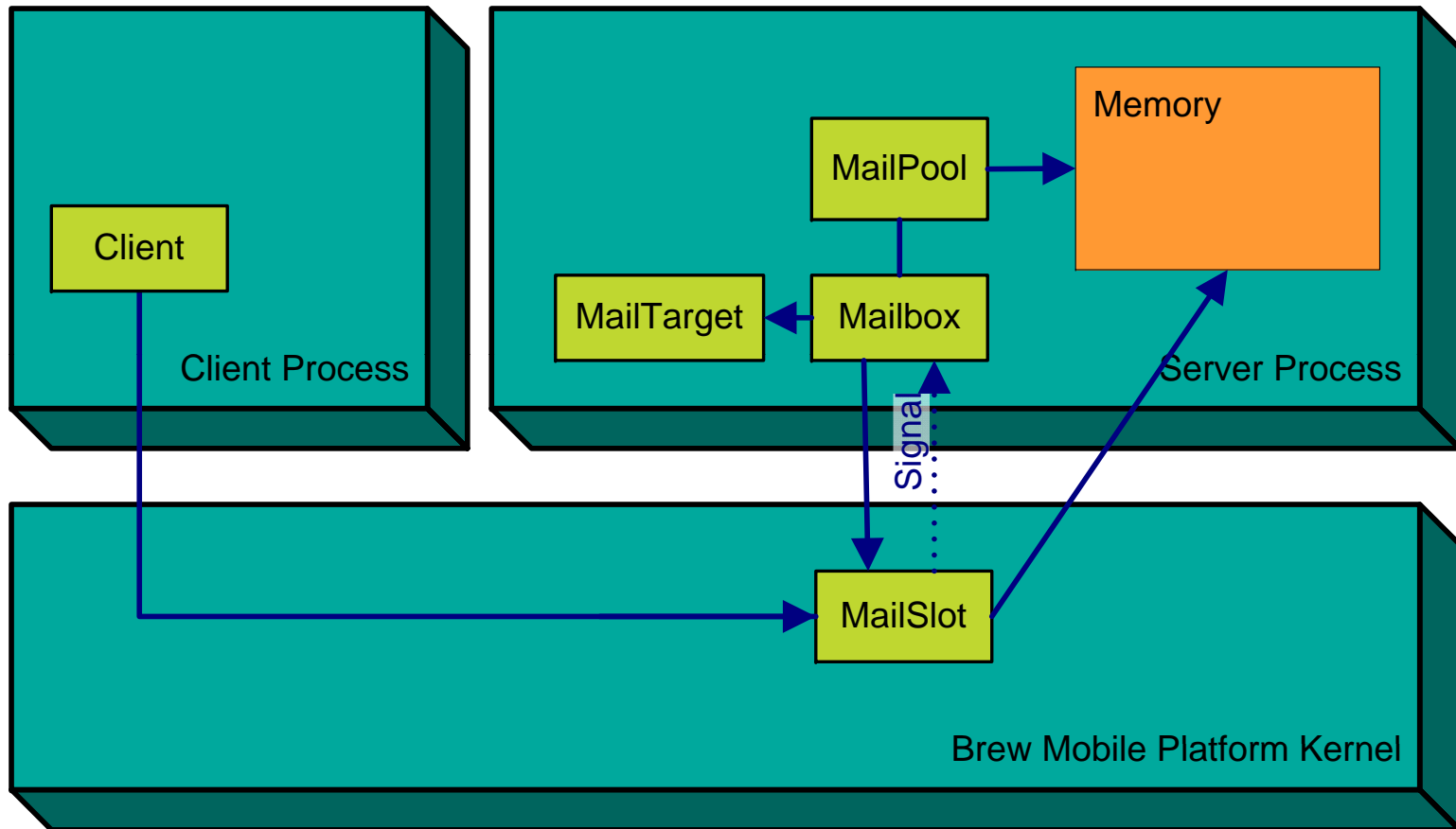
- Millisecond timer
 - ISysTimer Interface
 - `ISysTimer_AssociateSignal()` takes a signal and notifies when timer expires
- Finer granularity timers may be available



Utilities

- Enable thread-safe service implementations
 - Atomic Operations
 - Add
 - Exchange
 - Compare&Exchange
- Local Storage
 - Thread Local
 - Process Local
- Debug Utilities
 - Messages
 - Breakpoints

Asynchronous Messaging





Questions?