Advanced
Mac OS X Rootkits

Dino Dai Zovi
Chief Scientist
Endgame Systems

Overview

ENDGAME SYSTEMS

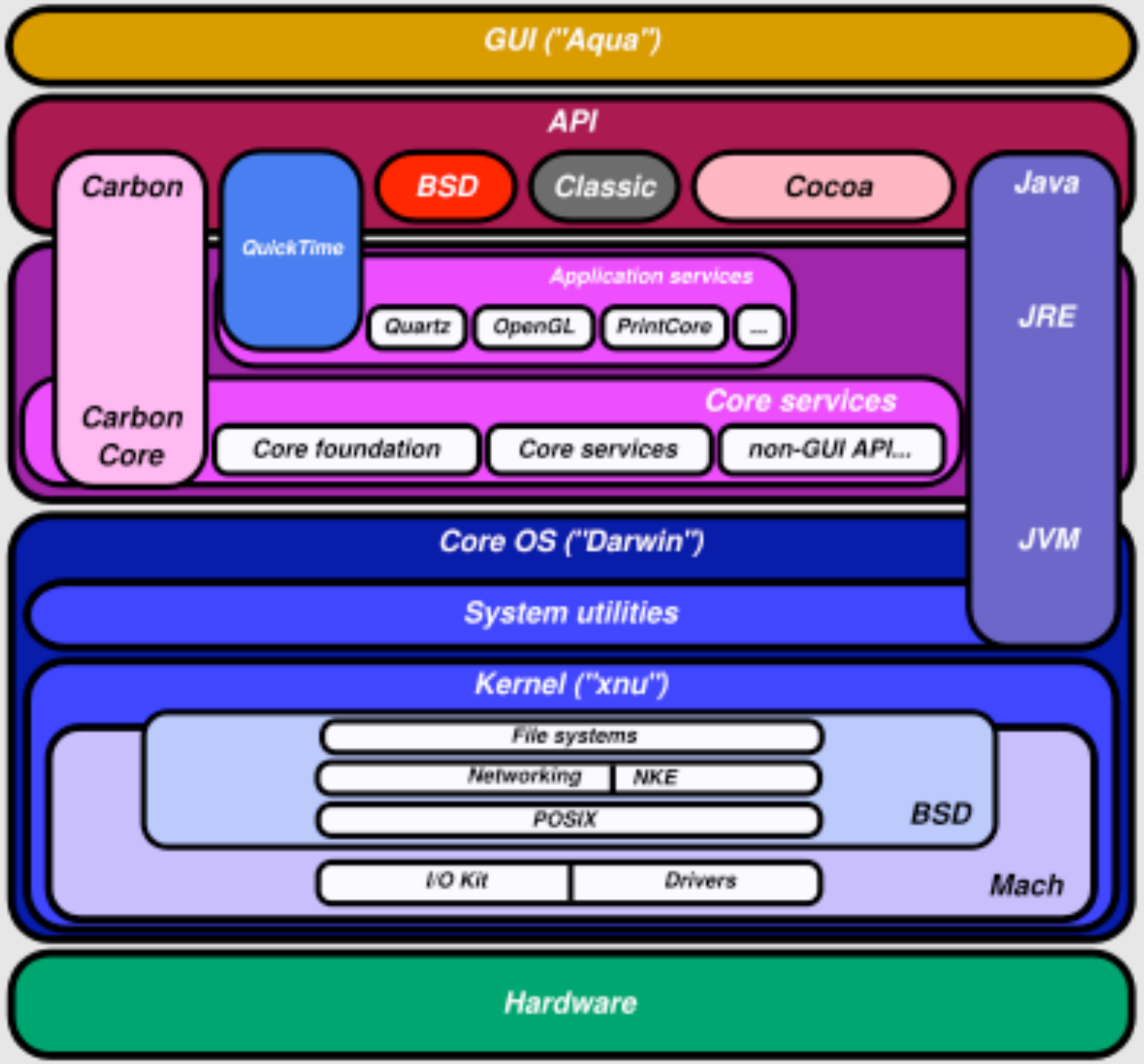
- Mac OS X and Mach
- Why use Mach for rootkits?
- User-mode Mach rootkit techniques
- Kernel Mach rootkit techniques

ENDGAME SYSTEMS

WHAT IS MACH?

Mac OS X System Architecture

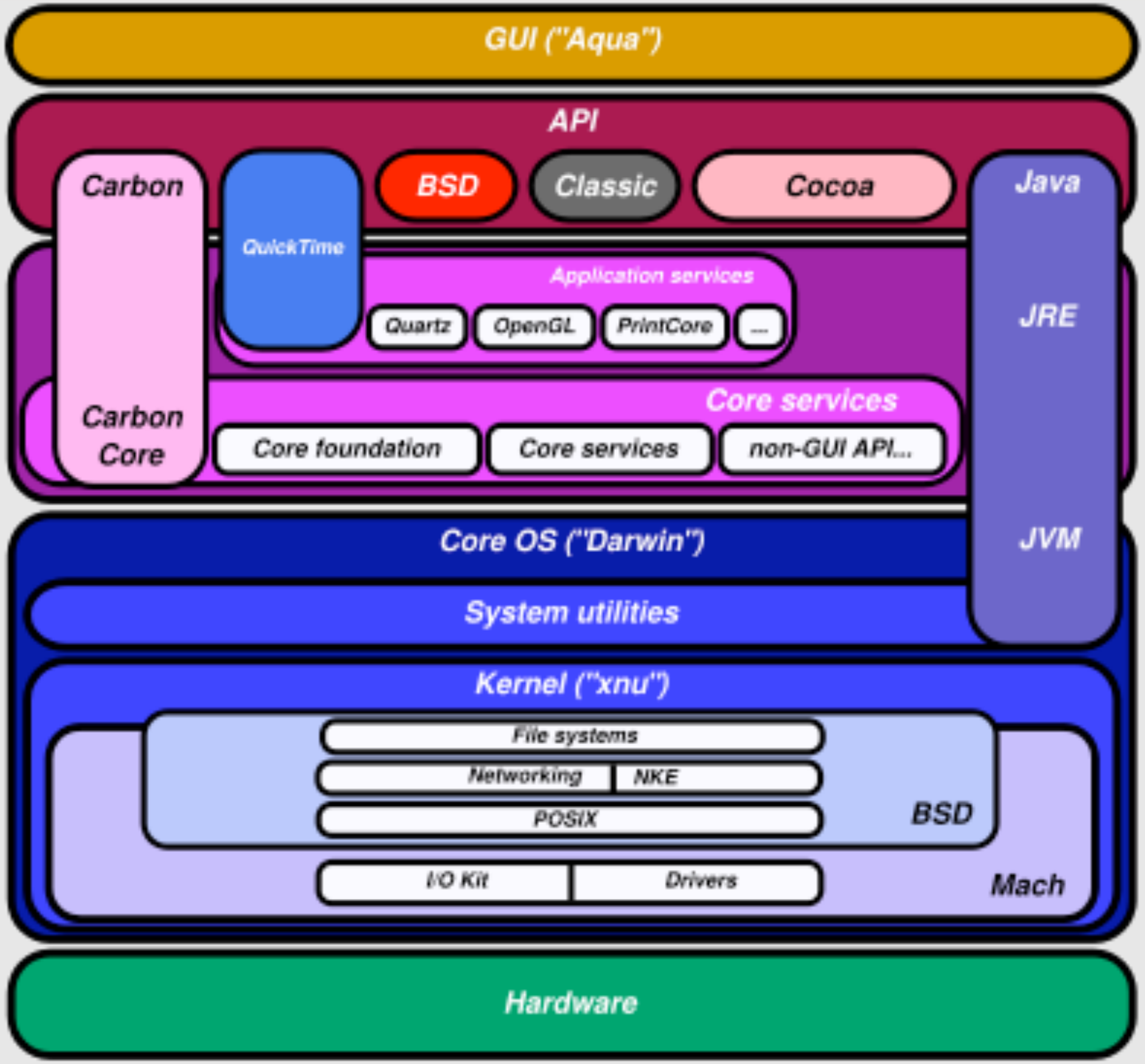
ENDGAME SYSTEMS



Mac OS X System Architecture

ENDGAME SYSTEMS

SUP DAWG, WE HEARD YOU LIKE KERNELS, SO WE PUT A KERNEL IN YOUR MICRO-KERNEL (SO YOU CAN MACH WHILE YOU BSD)



Introduction to Mach

ENDGAME SYSTEMS

- Mac OS X kernel (xnu) is a hybrid between Mach 3.0 and FreeBSD
  - FreeBSD kernel top-half runs on Mach kernel bottom-half
  - Multiple system call interfaces: BSD (positive numbers), Mach (negative)
  - BSD sysctls, ioctls
  - Mach in-kernel RPC servers, IOKit user clients, etc

Mach Microkernel Abstractions

ENDGAME SYSTEMS

- Task: A resource container
  - Virtual memory address space
  - 1 or more Threads
  - IPC port rights
- Thread: An entity that can be scheduled by the kernel to run on a processor
- Port: An inter-task communication mechanism using structured, reliable messages
  - Think of them as sender-restricted “P.O. Boxes”
  - Only exist in kernel, Tasks hold Port Rights
- Message: Data communicated between ports

Port Names and Rights

ENDGAME SYSTEMS

- A Port Name is a Task-specific 32-bit number that refers to a given port
  - Similar to file descriptors and Handles
- A Task holds unidirectional Port Rights that determine whether they may send and or receive messages on a Port
- Tasks may transfer Port Rights between each other by sending them in messages
  - Kernel transfers rights and allocates a new name in the receiving task

Example: Bootstrap server

ENDGAME SYSTEMS

- Tasks are created with a set of initial ports, one is the bootstrap port, Bootstrap server is the glue that allows different tasks to send messages to each other
- Task X registers a port with Bootstrap server under a string name (“foo”) by sending message on Bootstrap port
  - Transfers SEND rights to Bootstrap server
- Task Y looks up “foo” on bootstrap server by sending message to bootstrap port, replies with server port
  - Reply transfers SEND rights to Task Y
- Task Y can now send messages to Task X over that port

Mach Security Model

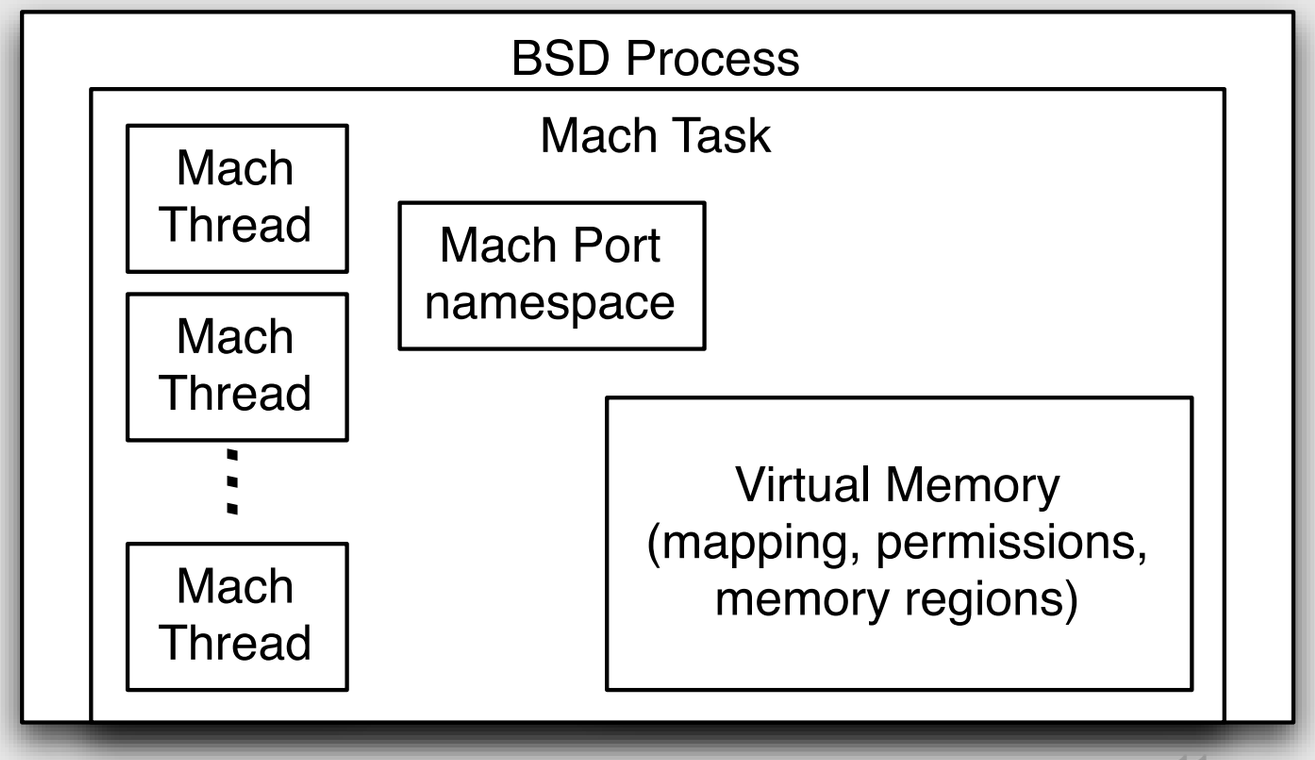
ENDGAME SYSTEMS

- Tasks, ports, and port rights form Mach's capability-based security model
- Mach has no notion of users or groups
- Obtaining SEND rights on a port may be a privilege escalation
  - SEND rights on another task's task port gives full control over that task

Mach Tasks vs. BSD Processes

ENDGAME SYSTEMS

- Mach Tasks own Threads, Ports, and Virtual Memory
- BSD Processes own file descriptors, etc.
- BSD Processes <=> Mach Task
  - task_for_pid(), pid_for_task()
- POSIX Thread != Mach Thread
  - Library functions use TLS



Mach RPC

ENDGAME SYSTEMS

- Mach RPC uses Mach messages for communication, NDR for data packing
- Interface files (.defs) are compiled by MiG into client and server stubs
- Most Mach kernel services are offered over RPC
  - i.e. thread, task, and VM control
- Mach kernel traps are minimal (it's a microkernel, remember?)

Mach Task/Thread System Calls

ENDGAME SYSTEMS

- Mach kernel RPC calls:
  - task_create(parent_task, ledgers, ledgers_count, inherit_memory, *child_task)
  - thread_create(parent_task, *child_activation)
  - vm_allocate(task, *address, size, flags)
  - vm_deallocate(task, address, size)
  - vm_read(task, address, size, *data)
  - vm_write(task, address, data, data_count)

ENDGAME SYSTEMS

MACH-BASED ROOTKITS

Why Mach-based Rootkits?

ENDGAME SYSTEMS

- Traditional Unix rootkit techniques are well understood
- Mach functionality is more obscure
- Rootkits using obscure functionality are less likely to be detected or noticed
- Mach is fun to program
  - Imagine re-learning Unix all over again

User-mode Mach Rootkits

ENDGAME SYSTEMS

- Not as “sexy” as kernel mode rootkits
- Can be just as effective and harder to detect
- Are typically application/process -specific
- Based on thread injection or executable infection
- Would you notice an extra bundle and thread in your web browser?

Injecting Mach Threads

ENDGAME SYSTEMS

- Get access to another task's task port
  - task_for_pid() or by exploiting a local privilege escalation vulnerability
- Allocate memory in remote process for thread stack and code trampoline
- Create new mach thread in remote process
  - Execute trampoline with previously allocated thread stack segment
  - Trampoline code promotes Mach Thread to POSIX Thread
    - Call _pthread_set_self(pthread_t) and cthread_set_self(pthread_t)

Mach Exceptions

ENDGAME SYSTEMS

- Tasks and Threads generate exceptions on memory errors
- Another thread (possibly in another task) may register as the exception handler for another thread or task
- Exception handling process:

1. A Thread causes a runtime error, generates an exception
2. Exception is delivered to thread exception handler (if exists)
3. Exception is delivered to task's exception handler (if exists)
4. Exception converted to Unix signal and delivered to BSD Process

Injecting Mach Bundles

ENDGAME SYSTEMS

- Inject threads to call functions in the remote process
  - Remote thread calls injected trampoline code and then target function
  - Function returns to chosen bad address, generates an exception
  - Injector handles exception, retrieves function return value
- Call dlopen(), dlsym(), dlclose() to load bundle from disk
- Inject memory, call NSCreateObjectFileImageFromMemory(), NSLinkModule()
- Injected bundle can hook library functions, Objective-C methods

inject-bundle

ENDGAME SYSTEMS

- inject-bundle
  - Inject a bundle from disk into a running process
  - Usage: inject_bundle path_to_bundle [pid]
- Sample bundles
  - test: Print output on load/run/unload
  - isight: Take a picture using iSight camera
  - sslspy: Log SSL traffic sent through SecureTransport
  - ichat: Log IMs from within iChat

Hooking and Swizzling

ENDGAME SYSTEMS

- Hooking C functions is basically the same as on any other platform
  - see Rentzsch's mach_override
- Objective-C runtime has hooking built-in:
  - method_exchangeImplementations()
  - or just switch the method pointers manually
  - all due to Obj-C's dynamic runtime
  - use JRSwizzle for portability

Rootkitting the Web Browser

ENDGAME SYSTEMS

- What client system doesn't have the web browser open at all times?
- Will be allowed to connect to *:80 and *:443 by host-based firewalls (i.e. Little Snitch)
- Injected bundles do not invalidate dynamic code signatures (used by Keychain, etc)

ENDGAME SYSTEMS

INJECTED BUNDLE DEMO

ENDGAME SYSTEMS

MACHIAVELLI

NetMessage and NetName servers

ENDGAME SYSTEMS

- Network transparency of IPC was a design goal
- Old Mach releases included the NetMessage Server
  - Mach servers could register themselves on the local NetName server
  - Clients could lookup named servers on remote hosts
  - Local NetMessage server would act as a proxy, transmitting Mach IPC messages over the network
- These features no longer exist in Mac OS X
- Machiavelli adds them back

Machiavelli

ENDGAME SYSTEMS

- Mach RPC provides high-level remote control
  - vm_alloc(), vm_write(), thread_create() on kernel or any task
- We want to still use MiG generated client RPC stubs, don't want to re-implement
- Machiavelli acts as a Mach RPC "bridge", allowing tasks on two different machines to do RPC between them (both directions)

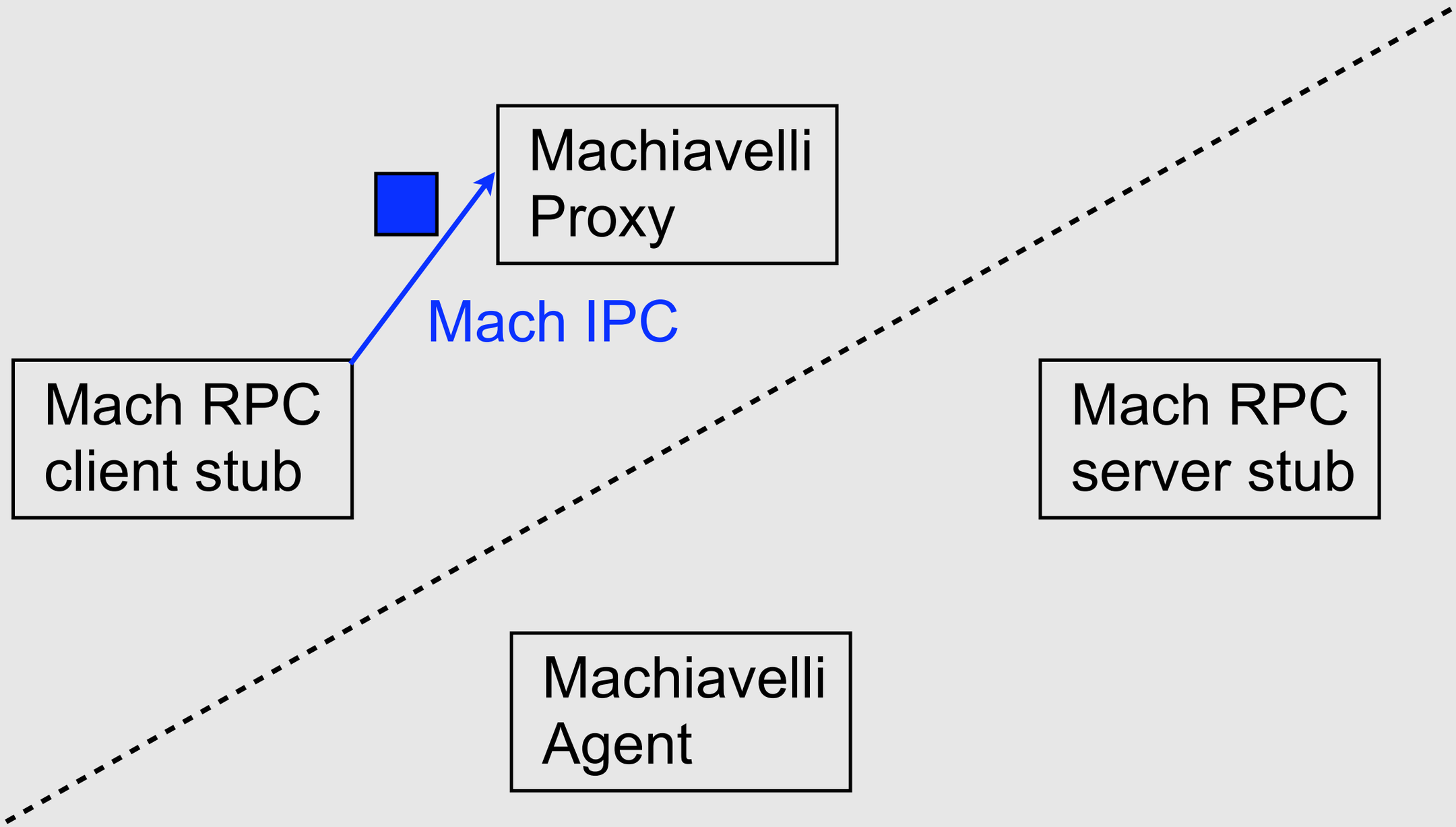
Machiavelli Architecture

ENDGAME SYSTEMS

- Machiavelli Proxy
  - Receives messages on proxy ports and sends to remote Agent
  - Replaces port names in messages received from remote Agent with proxy ports
- Machiavelli Agent
  - Receives messages over network from Proxy, sends to real local destination
  - Receives and transmits reply message if a reply is expected
- Machiavelli RPC Server
  - Provides miscellaneous “glue” functionality like task_for_pid(), sysctl(), etc.

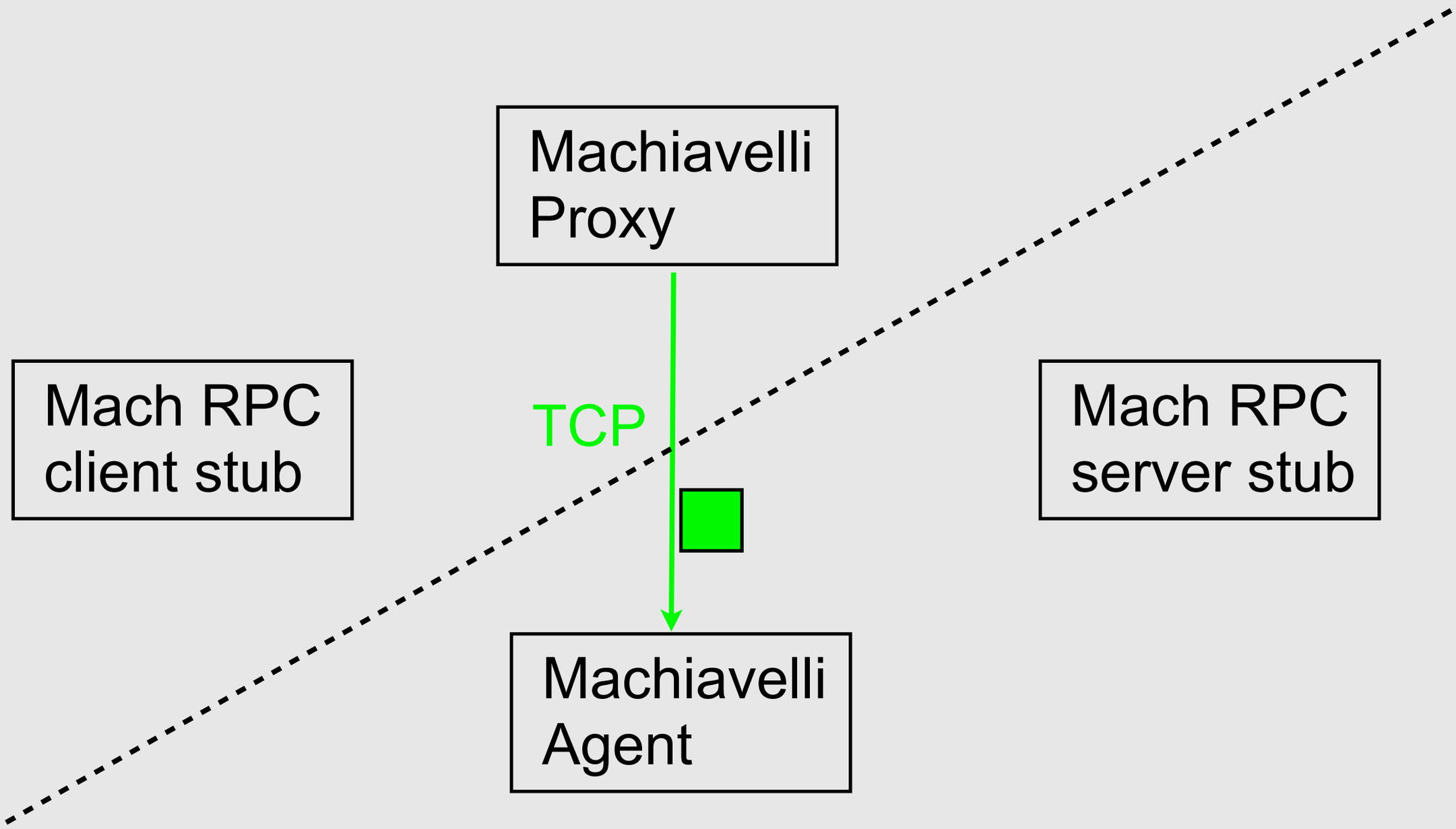
Machiavelli Message Flow

ENDGAME SYSTEMS



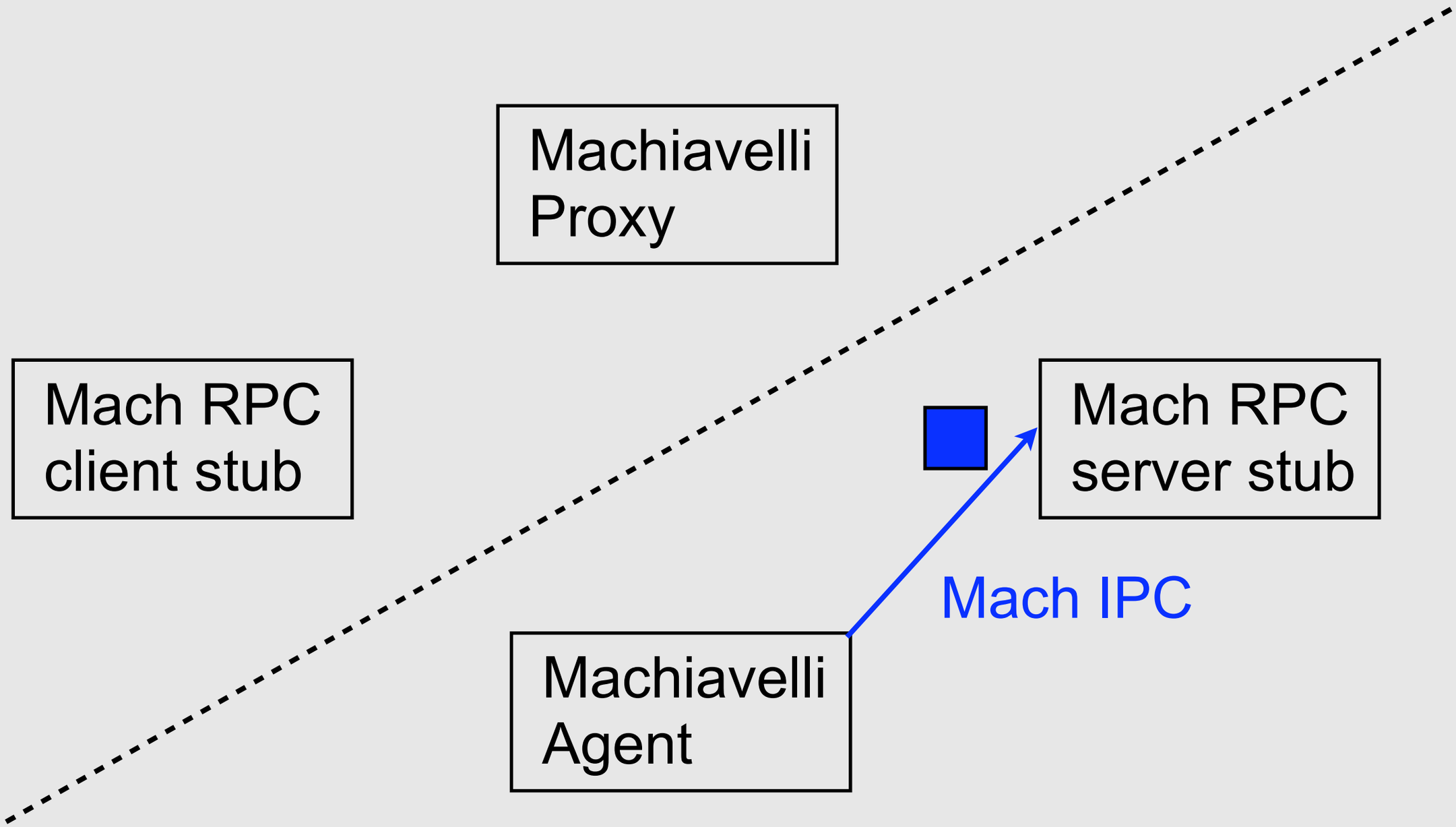
Machiavelli Message Flow

ENDGAME SYSTEMS



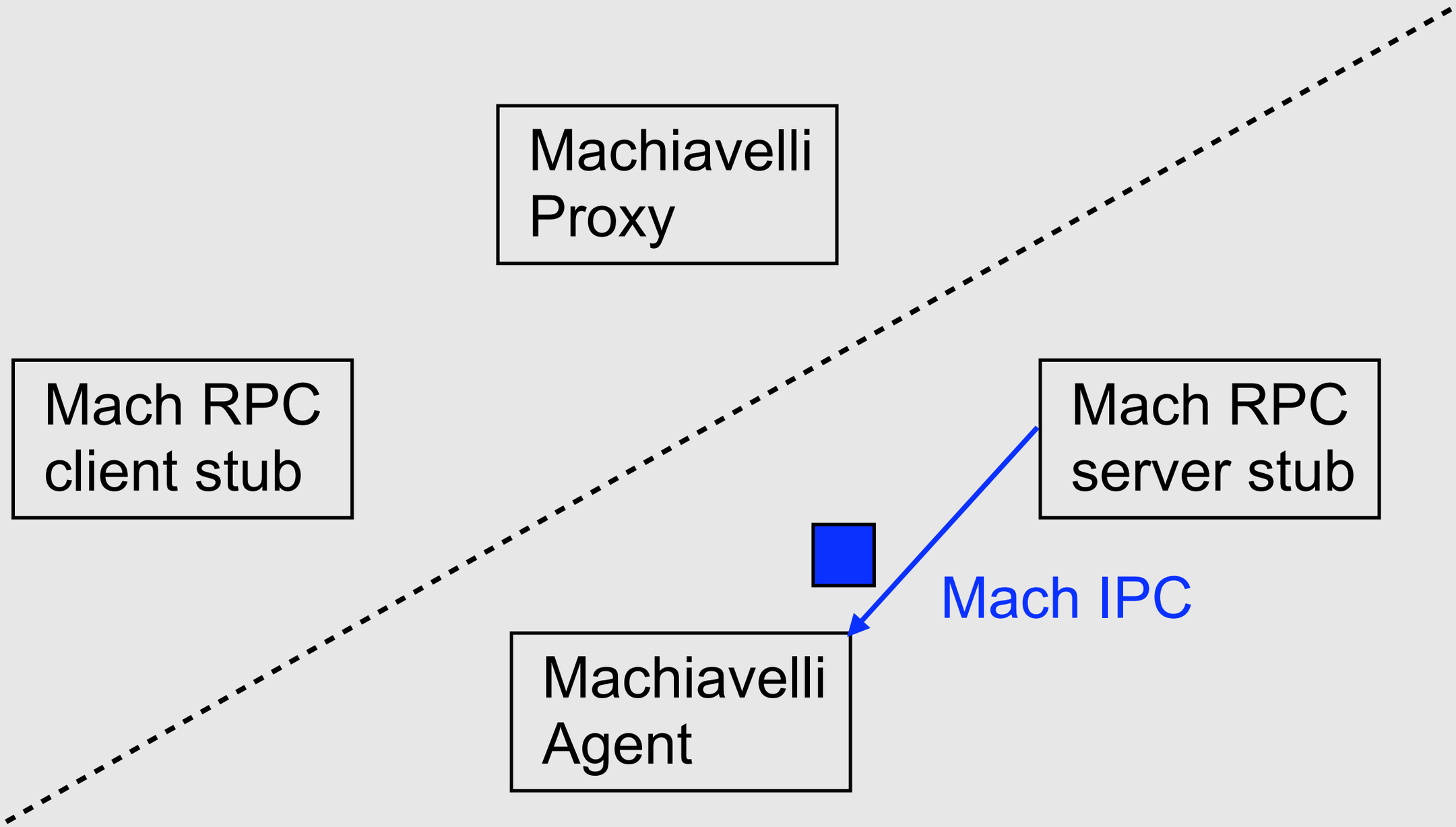
Machiavelli Message Flow

ENDGAME SYSTEMS

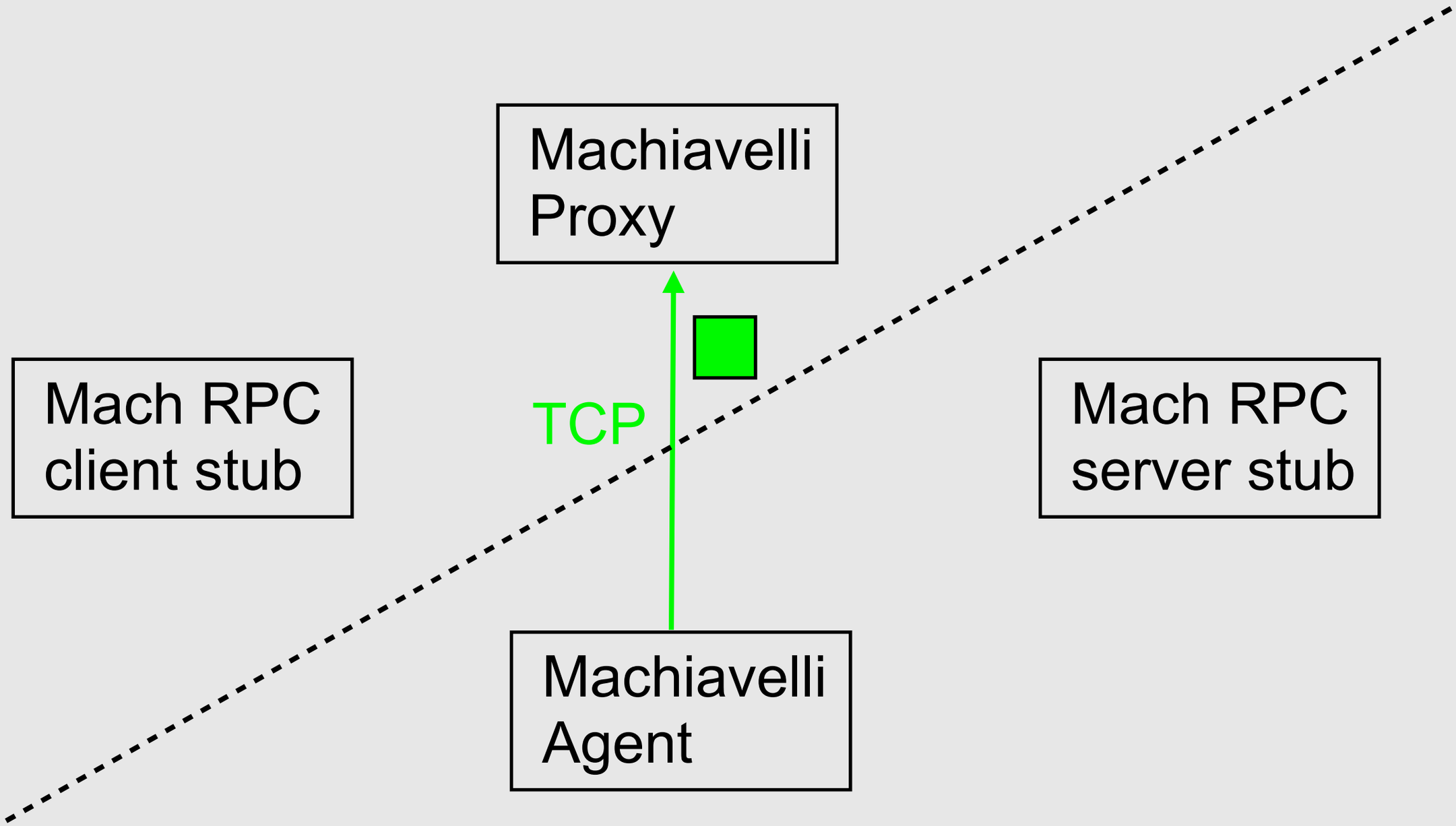


Machiavelli Message Flow

ENDGAME SYSTEMS

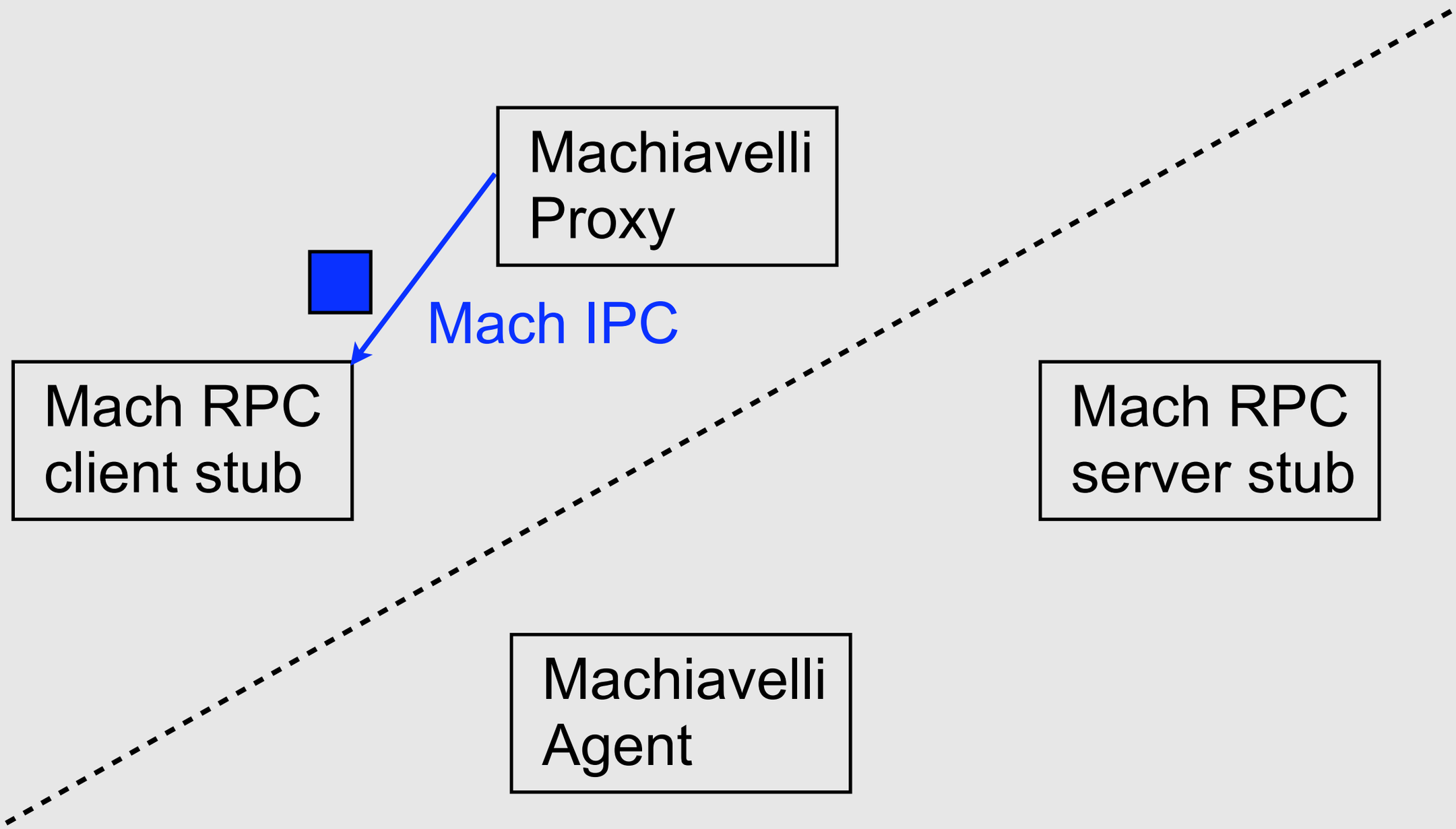


Machiavelli Message Flow



Machiavelli Message Flow

ENDGAME SYSTEMS



Mach messages

ENDGAME SYSTEMS

- Mach messages are structured and unidirectional
- Header:

```
typedef struct
{
  mach_msg_bits_t       msgh_bits;
  mach_msg_size_t       msgh_size;
  mach_port_t           msgh_remote_port;
  mach_port_t           msgh_local_port;
  mach_msg_size_t       msgh_reserved;
  mach_msg_id_t         msgh_id;
} mach_msg_header_t;
```

- Body consists of typed data items

Complex Mach Messages

ENDGAME SYSTEMS

- “Complex” Mach messages contain out-of-line data and may transfer port rights and/or memory pages to other tasks
- In the message body, descriptors describe the port rights and memory pages to be transferred
- Kernel grants port rights to the receiving process
- Kernel maps transferred pages to receiving process, sometimes at message-specified address

Proxying Mach Messages

ENDGAME SYSTEMS

- Proxy maintains a Mach port set
  - A port set has the same interface as a single port and can be used identically in mach_msg()
  - Each proxy port in the set corresponds to the real destination port name in the remote Agent
  - Port names can be arbitrary 32-bit values, so port set names are pointers to real destination port name values
- Received messages must be translated (local <=> remote ports and descriptor bits)
- Messages are serialized to byte buffers and then sent to Agent

Serializing Mach Messages

ENDGAME SYSTEMS

- Serializing “simple” messages is simple as they don’t contain any out-of-line data
- Out-of-line data is appended to the serialized buffer in order of the descriptors in the body
- Port names are translated during deserialization
  - Translating to an intermediate “virtual port name” might be cleaner

Deserializing Mach Messages

ENDGAME SYSTEMS

- Port names in the mach message must be replaced with local port names
- On Agent, this is done to receive the reply
- On Proxy, this is done to replace transferred port names with proxy port names
  - Ensures that only the initial port must be manually obtained from the proxy, the rest are handled automatically
- OOL memory is mapped+copied into address space

Machiavelli example


```
int main(int argc, char* argv[])
{
    kern_return_t kr;
    mach_port_t port;
    vm_size_t page_size;

    machiavelli_t m = machiavelli_init();
    machiavelli_connect_tcp(m, "192.168.13.37", "31337");
    port = machiavelli_host_self(m);

    if ((kr = _host_page_size(port, &page_size)) != KERN_SUCCESS) {
        errx(EXIT_FAILURE, "_host_page_size: %s", mach_error_string(kr));
    }

    printf("Remote host page size: %d\n", page_size);

    return 0;
}
```

ENDGAME SYSTEMS

MACHIAVELLI DEMO

Miscellaneous Agent services

ENDGAME SYSTEMS

- Agent must provide initial Mach ports:
  - host port
  - task_for_pid() (if pid == 0 => returns kernel task port)
- As OS X is a Mach/Unix hybrid, just controlling Mach is not enough
  - i.e. How to list processes?
- Instead of implementing Unix functionality in Agent, inject Mach RPC server code into

Network Kernel Extensions (NKEs)

ENDGAME SYSTEMS

- NKEs can extend or modify kernel networking functionality via:
  - Socket filters
  - IP filters
  - Interface filters
  - Network interfaces
  - Protocol plumbers

ENDGAME SYSTEMS

MACH IN THE KERNEL

Mac OS X Kernel Rootkits

ENDGAME SYSTEMS

- Most Mac OS X Kernel Rootkits do traditional Unix-style syscall filtering
- They load as kernel extensions and remove themselves from kmod list
  - WeaponX, Mac Hacker's Handbook, Phrack 66
  - Hides the kernel extension and prevents removal
- Alternatively, code can be directly loaded into kernel via vm_allocate(), and vm_write()
- In both cases, we have an unauthorized Mach-O object file in the kernel

Uncloaking Kernel Rootkits

ENDGAME SYSTEMS

- Access to kernel task exposes kernel memory regions map and gives direct access to kernel memory
  - get kernel task via task_for_pid(0)
- Iterate over allocated regions, examining beginning for Mach-O headers
  - Avoid reading volatile memory, causes panic
- Multiple Mach-O headers will may be found pointing to the same text and data segments
- Compare identified segments to segments loaded by kernel extensions in the kmod list
- Suspicious Mach-O objects can be dumped to disk for reverse engineering and analysis

ENDGAME SYSTEMS

UNCLOAK DEMO

Evasive Maneuvers

ENDGAME SYSTEMS

- Unix system call filtering can't evade Mach kernel RPC (different interface)
- There's no reason why kernel rootkits need to be loaded as Mach-O objects
  - vm_allocate() + thread_create() on kernel task
  - DKOM, return-oriented rootkits, etc.
- Alternatively, filter in-kernel Mach RPC servers

Interposing on Kernel Mach RPC

ENDGAME SYSTEMS

- Mach system calls allow Mach RPC to in-kernel servers which perform task, thread, and VM operations
- RPC routines are stored in the `mig_buckets` hash table by subsystem id + subroutine id
- Analogous to `sysent` table for Unix system calls
- Incoming Mach messages sent to a kernel-owned port are dispatched through `mig_buckets`
- We can interpose on these function calls or inject new RPC servers by modifying this hash table

Example: inject_subsystem


```
int inject_subsystem(const struct mig_subsystem * mig)
{
        mach_msg_id_t h, i, r;
        // Insert each subroutine into mig_buckets hash table
        for (i = mig->start; i < mig->end; i++) {
                mig_hash_t* bucket;
                h = MIG_HASH(i);
                do { bucket = &mig_buckets[h % MAX_MIG_ENTRIES];
                } while (mig_buckets[h++ % MAX_MIG_ENTRIES].num != 0 &&
                                h < MIG_HASH(i) + MAX_MIG_ENTRIES);
                if (bucket->num == 0) {     // We found a free spot
                        r = mig->start - i;
                        bucket->num = i;
                        bucket->routine = mig->routine[r].stub_routine;
                        if (mig->routine[r].max_reply_msg)
                                bucket->size = mig->routine[r].max_reply_msg;
                        else
                                bucket->size = mig->maxsize;
                        return 0;
                }
        }
        return -1;
}
```

Mach Kernel RPC servers

ENDGAME SYSTEMS

- In-kernel Mach RPC subsystems are enumerated in the `mig_e` table and interfaces are in /usr/include/mach/*subsystem.defs*
  - mach_vm, mach_port, mach_host, host_priv, host_security, clock, clock_priv, processor, processor_set, is_iokit, memory_object_name, lock_set, ledger, semaphore, task, thread_act, vm_map, UNDReply, default_pager_object, security

Conclusion

ENDGAME SYSTEMS

- Mach is a whole lot of fun
- Mach IPC can be made network transparent and provides a good abstraction for remote host control
- I wish my desktop OS was as secure as iPhone OS
- For updated slides and tools go to:
  - http://trailofbits.com/