SHARING DATA IN MULTI-PROCESS APPLICATIONS

Professor Ken Birman
CS4414 Lecture 18

IDEA MAP FOR TODAY

Complex Systems often have many processes in them. They are not always running on just one computer.

Modern solutions of this kind often need to run on clusters of computers or in the cloud, and need sharing approaches that work whether processes are local (same machine) or remote.

Linux offers too many choices! They include pipes, mapped files (shared memory), DLLs.
Linux weakness: the "single machine" look and feel.

As a developer, you think of the cloud itself as a kind of distributed operating system kernel, offering tools that work from "anywhere".

LARGE, COMPLEX SYSTEMS

Large systems often involve multiple processes that need to share data for various reasons.

Components may be in different languages: Java, Python, C++, O'CaML, etc…

Big applications are also broken into pieces for software engineering reasons, for example if different teams collaborate

MODERN SYSTEMS DISTINGUISH TWO CASES

Many modern systems use “standard libraries” to interface to storage systems, or for other system services.

You think of the program as an independent agent, but it uses the same library as other programs in the application.

Here, the focus is on how to build libraries that many languages can access. C++ is a popular choice.

LOCAL OPTIONS

These assume that the two (or more) programs live on the same machine.

They might be coded in different languages, which also can mean that data could be represented in memory in different ways (especially for complicated objects or structures – but even an integer might have different representations!)

SINGLE ADDRESS SPACE, TWO (OR MORE) LANGUAGES

Issue: They may not use the same data representations!

JAVA NATIVE INTERFACE

The Java Native Interface (JNI) allows Java applications to talk to libraries in languages like C or C++.

In effect, you build a Java "wrapper" for each library method.

JNI will load the C++ DLL at runtime and verify that it has the methods you expected to find.

JNI DATA TYPE CONVERSIONS

JNI has special accessor methods to access data in C++, and then the wrapper can create Java objects that match.

For some basic data types, like int or float, no conversion is needed.   For complex ones, where conversion does occur, the cost is similar to the cost of copying.

JNI is generally viewed as a high-performance option

FORTRAN CAN EASILY “TALK” TO C++

Fortran is a very old language, and the early versions made memory structs visible and very easy to access.

This is still true of modern Fortran: the language has evolved enormously, but it remains easy to talk to “native” data types.

So Fortran to C++ is particularly effective.

PYTHON IS TRICKY

There are many Python implementations.

The most widely popular ones are coded in C and can easily interface to C++. There are also versions coded in Java, etc.

But because Python is an interpreter, Python applications can’t just call into C++ without a form of runtime reflection.

HOW PYTHON FINESSES THIS

Python is often used control computations in “external” systems.

For example, we could write Python code to tell a C++ library to load a tensor, multiply it by some matrix, invert the result, then compute the eigenvalues of the inverted matrix…

The data could live entirely in C++, and never actually be moved into the Python address space at all! Or it could even live in a GPU

PYTHON INTEGERS

One example of why it isn't so trivial to just share data is that Python has its own way of representing strings and even integers

A Python integer will use native representations and arithmetic if the integer is small. But Python automatically switches to a larger number of bits as needed and even to a Bignum version.

So… if Python wants to send an integer to C++, we run into the risk that a C++ integer just can't hold the value!

SOLUTION? USE “BINDINGS”

Boost.Python leverages this basic mechanism to let you call Python from C++ or C++ from Python.

1) You need to create a plain C (not C++) “interface” layer. These methods can only take native data types + pointers.

2) Compile it and create a DLL. In Python, load this DLL, then import the interface methods.

4) Now you can call those plain C methods, if you follow certain (well-documented) rules (like: no huge integers!). To call an object instance method, you pass a pointer to the object and then the arguments, as if “this” was a hidden extra argument.

SHARING WITH DIFFERENT PROCESSES

Issue: They have different address spaces!

SHARING BETWEEN DIFFERENT PROCESSES

Large multi-component systems that explicitly share objects from process to process need tools to help them do this.

Unlike language-to-language, the processes won't be linked together into a single address space.

Because *cloud computing* is so popular, these tools often are designed to work over a network, not just on a single NUMA computer.

IF PROCESSES ARE ON A SINGLE (NUMA) MACHINE, WE HAVE A FEW "OLD" SHARING OPTIONS:

1. Single address space, threads share memory directly.
2. Linux pipes. Assumes a "one-way" structure.
3. Shared files. Some programs could write data into files; others could later read those files.
4. <u>Mapped</u> files. Same idea, but now the readers can instantly see the data written by the (single) writer. Also useful as a way to skip past the POSIX API, which requires copying (from the disk to the kernel, then from the kernel into the user's buffer).

DIMENSIONS TO CONSIDER

Performance, simplicity, security. Some methods have very different characteristics than others.

Ease of later porting the application to a different platform . Some modern systems are built as a collection of processes on one machine, but over time migrate to a cluster of computers.

Standardization. Whatever we pick, it should be widely used.

LET'S LOOK AT SOME EXAMPLES

The C++ command runs a series of sub-programs:

1. The "C preprocessor", to deal with #define, #if, #include
2. The template analysis and expansion stage
3. The compiler, which has a parsing stage, a compilation stage, and an optimization stage.
4. The assembler
5. The linker

… they share data by creating files, which the next stage can read

WHY DOES C++ USE FILE SHARING?

C++ was created as a multi-process solution for a single computer. In the old days we didn't have an mmap system call.

Also, since one process writes a file, and the next one reads it sequentially and "soon", after which it gets deleted, Linux is smart enough to keep the whole file in cache and might never even put it on disk.

There are many such examples on Linux. Most, like C++, have a controlling process that launches subprocesses, and most share files from stage to stage.

ANOTHER OPTION: MMAP THE FILES

We learned about mmap when we first saw the POSIX file system API. At one time people felt that mmap could become the basis for shared objects in Linux.

Linux allocates a segment of memory for the mapped file. Mmap returns the base address of this segment.

Idea: mmap a memory segment, then allocate objects in it.

A MAPPED FILE IS LIKE A BIG BYTE ARRAY

This is sometimes very convenient

If the data being shared is some form of raw information, like pixels in a video display, or numbers in a matrix, it works well.

There is a way to create a mapped file with no actual disk storage. This form of shared memory can be useful!

MAPPED FILES

Many Wall Street trading firms have real-time ticker feeds of prices for the stocks and bonds and derivatives they trade.

Often this is managed via a daemon that writes into a shared file. The file holds the history of prices.

By mapping the head of the file, processes can track updates. A library accesses the actual data and handles memory fencing.

SHARED MEMORY

Many gaming platforms use a set of processes that share memory via mapped files.

These systems disable the "storage" part of the mapped file, so no I/O occurs. They end up with a pure mapped "segment"

The advantage is that the game engine can be a separate process from the GUI.

SHARED MEMORY

We also use shared memory to access video displays.

- The hardware for modern screens is quite fancy.
- But basically, there is a mapped memory segment your application can access. It sends "commands" as a stream to a special CPU running a special video language. It may also leverage a GPU.
- However, and this is important, *there is no corresponding file on disk!*
- The benefit of shared memory is that data rates are too high to write this data into a file or send it over a pipe.

LINUX ITSELF USES MAPPED FILES

The DLL concept (“linking”) is based on a mapped file.

In that case the benefits are these:

- The file actually contains executable instructions. These must be in memory for the CPU to decode and execute.
- But the DLL can be shared between multiple applications, saving memory and improving L3 caching performance.

SHARING WITH DIFFERENT MACHINES

Issue: Now we need to also deal with the network

NETWORKED SETTINGS REQUIRE DIFFERENT APPROACHES

When we run in a networked environment, we need tools that will work seamlessly even if the processes are on different machines.

Mapped files or segments are single-machine solutions. Mmap can be made to work over a network, but performance is disappointing and this option is not common.

CLOUD COMPUTING

In other courses, you'll use modern cloud computing systems.

Those are like a large multicomputer kernel, with services that programs can use *no matter which machine they run on.*

Cloud computing has begun to reshape the ways we develop complex programs even on a single Linux machine.

DIFFERENT MACHINES + INTERNET

1. We will learn about TCP soon… like a pipe, but between machines. This extends the pipe option to the cloud case!
2. We could use a technique called "remote procedure call" where one process can invoke a method in a remote on. We will learn about this soon, too.
3. We could pretend that everything is a web service, and use the same tools that web browsers are built from.

AMAZON.COM

Prior to 2005, Amazon web pages were created by a single server per page. But these servers were just not fast enough.

Famous study: 100ms delay reduces profits by nearly 1%

Today, a request is handled by a "first tier" server supported by a collection of services (as many as 100 per page)

AMAZON INVENTED CLOUD COMPUTING!

The Amazon services are used by browsers from all over the world: a networked model.

And Amazon’s explicit goal was to leverage warehouses full of computers (modern “cloud computing” data centers).

… So Amazon is a great example of a solution that needs to use networking techniques.

INSIDE THE CLOUD?

Users of cloud computing platforms like Amazon's AWS, Microsoft's Azure, or Google Cloud don't need to see the internals.

They see a file system that is available everywhere, as well as other kernel services that look the same from every machine.

The individual machine runs Linux, yet these services make it very easy to spread one application over multiple machines!

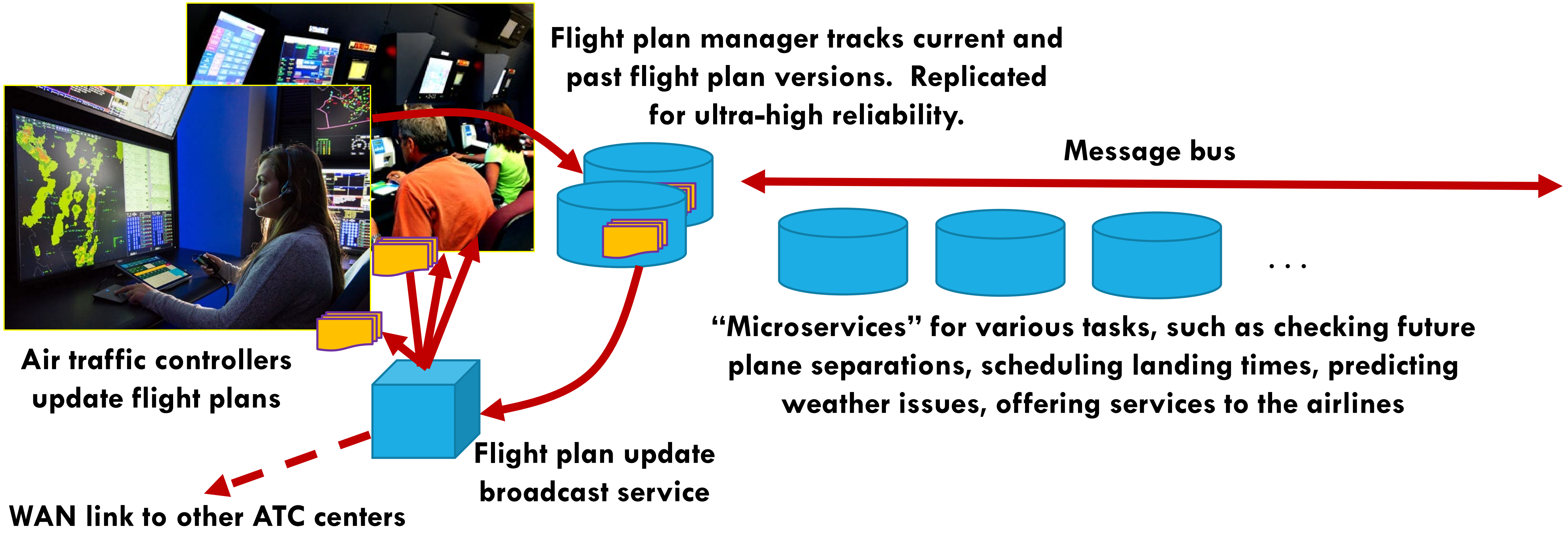
AIR TRAFFIC CONTROL

Ken worked on the French ATC solution

This system has been continuously used since 1996. It runs on a private cloud, but uses cloud-computing ideas.

ATC systems have many modules that cooperate. The “flight plan” is the most important form of shared information.

AIR TRAFFIC CONTROL SYSTEM


Flight plan manager tracks current and past flight plan versions. Replicated for ultra-high reliability.
Message bus
. . .
"Microservices" for various tasks, such as checking future plane separations, scheduling landing times, predicting weather issues, offering services to the airlines
Air traffic controllers update flight plans
Flight plan update broadcast service
WAN link to other ATC centers

SOFTWARE ENGINEERING AT LARGE SCALE

Big modern applications are created by software teams

They define modular components, which could co-exist in one address space or might be implemented by distinct programs

There is a science of *software engineering* that focuses on best ways of collaborating on big tasks of this kind.

SOFTWARE ENGINEERING AT LARGE SCALE

Each team needs a way to work independently and concurrently.

The teams agree on specifications for each component, then build, debug and unit test their component solutions.

We often pre-agree on some of the unit tests: “release validation” tests and “acceptance” tests. Integration occurs later when all the elements seem to be working.

SHOULD WE SHARE OBJECTS… OR FILES?

If we agree that component A will do something, then produce a file that becomes input to component B, and we agree on the file format and contents, the teams can already start work.

The A and B “interfacing” team would jointly construct some hand-crafted instances of the files A might output. Both teams check their solutions against these files.

FILES WORK IN ALL SETTINGS

Up to now we have always used the “local” file system on our Linux machines.

But Linux can also access a “remote” file system, and these can be shared by many machines.

So sharing via files works at any scale.

ADVANTAGES OF FILES

The B component team can run their solution again and again with the identical inputs.

This facilitates debugging and is a valuable form of unit test.

If the test files are complete, most of the B functionality gets checked.

DISADVANTAGES OF FILES

Files need to be read block by block.

Perhaps A works with “objects” and B is expected to treat them as objects. Yet the file will only contain bytes: the object format and layout is lost.

The file blocks might not correspond to any form of data chunks

MORE DISADVANTAGES

In Linux, temporary files are very common and can be inefficient:

- Editors write the whole new version of your file to disk, sync the file (to be sure it is actually on the disk), then use a file rename operation to “atomically” replace the old version.
- C++ stages use files to pass intermediary information
- Many applications have lock files, used very briefly.

Issue: The file “lifetime” might be just a few milliseconds!

MORE DISADVANTAGES

In Linux, temporary files are very common and can be inefficient:

This issue was noticed by researchers about 15 years ago.

Linux was modified to not actually write the data out, if permitted, and also to cache entire recently-written files in the kernel disk buffer, just in case it will be read immediately after creation.

But some applications like databases and the editor actually need to be sure the temporary file was written to disk – this is called "write-ahead logging" or "write-ahead file storage" and provides crash-tolerance guarantees. Those can't avoid the overheads of the disk I/O

Iss

MULTI-LINGUAL ISSUE

Modularity permis us to use different languages for different tasks. For example, a great deal of existing ATC code is in Fortran 77.

Byte arrays (or text files, character strings) are a least common denominator. Every language has a way to easily access them.

Modern systems have converged around the idea that this matches best with some form of "message passing".

SERIALIZATION/DESERIALIZATION

Converting an object to a byte array serializes the object. Later we deserialize to recreate the object.

A serialized object can be stored in a file, or we can use a "message passing" technology to send them from process to process over a network.

FEATURES OF SERIALIZATION TECHNOLOGIES

Some have notions of *software version numbers*. These allow you to ensure that software is properly patched and upgraded.

It is unwise to pass an object from version 2.0 of some component to version 1.0 of the next component. This mix might never have been tested!

FULLY ANNOTATED OBJECTS?

In addition to version numbering, it is important to document the data types in use, sizes of arrays, requirements or assumptions that methods are making, limits on sizes of things, permissions required, etc.

It is easy to “serialize” an object into a byte-array format containing pure data. But there is very little agreement on how these annotation should look.

DATA REPRESENTATIONS AND PADDING

An additional issue is that computers and languages can use different representations.

For example, even on a single machine, some languages end character strings with a null byte (0). Others track the string length.

And if data is shared between machines, different computer vendors often use CPU chips that represent numbers in different ways!

DATA REPRESENTATION ISSUES

Each language represents objects in its own way.

For example, in Python every integer can have unlimited numbers of digits.

In C++, the various int types match hardware word sizes: 8, 16, 32, 64 and 128 bits. So there are Python integers that can't fit into any C++ data type, unless you use a Bignum package.

MULTI-LINGUAL APPLICATIONS

But shared segments are not popular for applications like the air traffic control system.

That sort of system often has components in C++, components in Java or Python, components in Fortran

How are objects like "flight plans" shared in such systems?

NETWORKING STANDARDS AND FLEXIBILITY

If we think about Linux pipes, they are extremely simple and flexible. The main cost is simply that the data itself is a byte stream.

Developers began to question all of these shared memory ideas and complexities. **Are they worth all the trouble?**

THREE EXAMPLES OF STANDARDS

CORBA: A standard architecture for sharing objects between programs or components from many languages or developers.

Google RPC (GRPC): A faster way for a client program to invoke a method in a server, perhaps over the Internet. We'll discuss this soon.

Web services: An approach in which web pages in HTML are used to share information between programs. Widely available but slow.

ROLE OF A STANDARD

Like POSIX, a standard specifies rules that vendors agree to respect, in their mutual interest.

Standards for object sharing allow different companies to build solutions that interoperate.

CORBA is the most widely used standard for encoding objects and later decoding them. In between, we have a byte array.

COST ANALYSIS EXAMPLE: AIR TRAFFIC FLIGHT PLAN IN THE ATC SYSTEM WE SAW

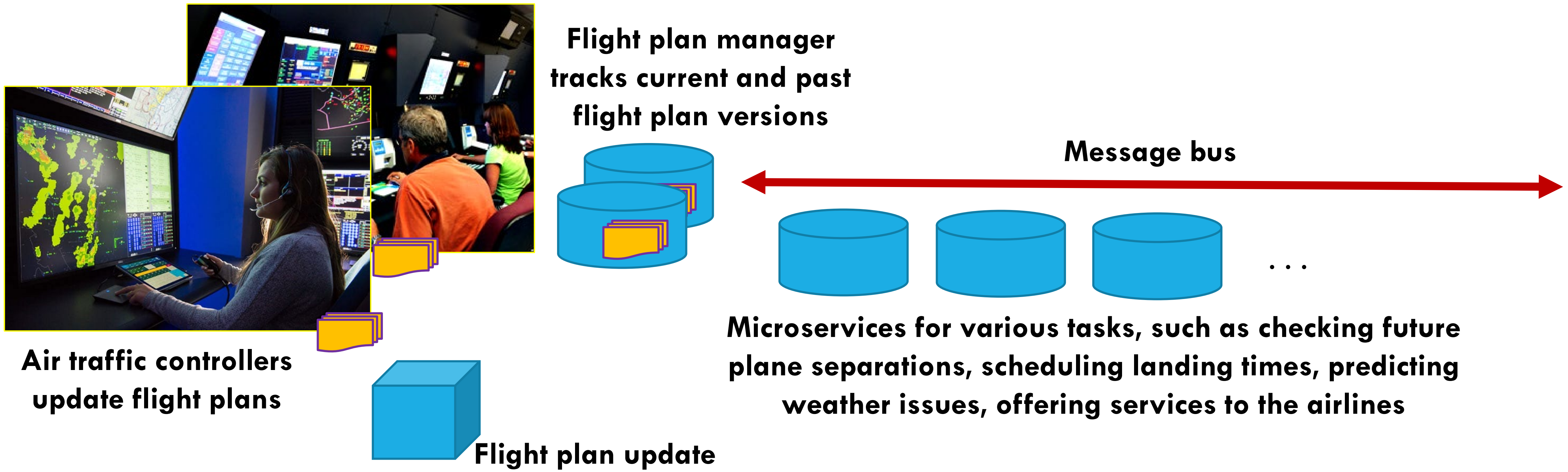
In memory, a flight plan is generally no more than 125k bytes.

With CORBA encoding, this grows to between 1MB and 10MB

- All numbers are "printed out", usually in base 10
- CORBA includes details on the way the data types were declared, version information, etc.

Effect? In some ATC settings, the system spends more time encoding and decoding flight plans than controlling aircraft!

WHERE ARE OBJECTS MOVED OR SHARED?



Flight plan manager tracks current and past flight plan versions

Message bus

. . .

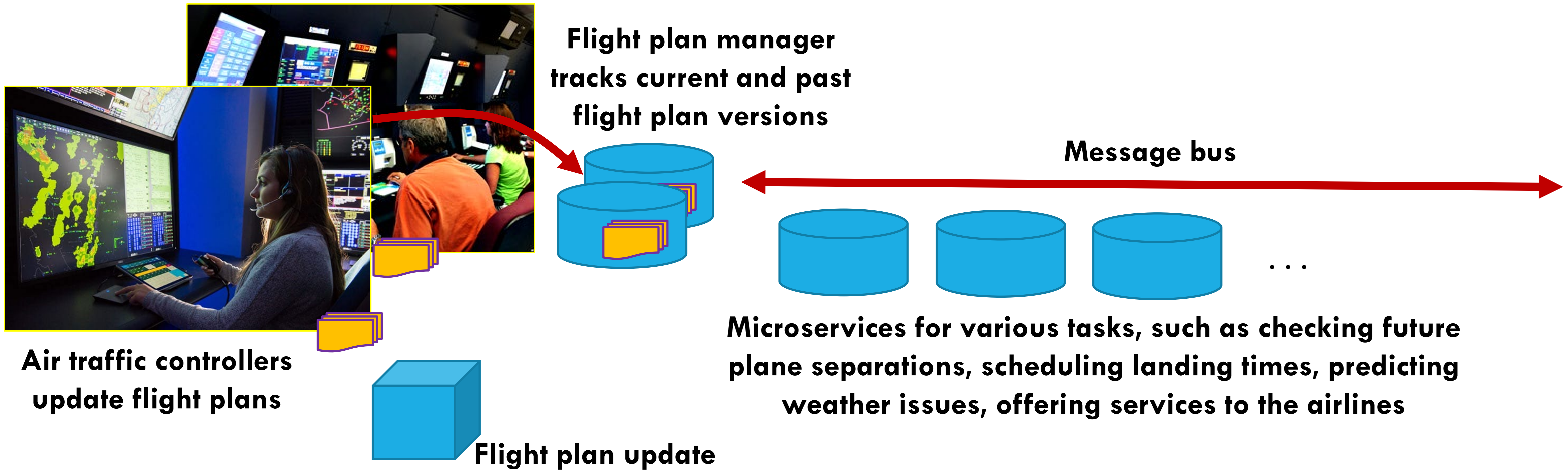
Microservices for various tasks, such as checking future plane separations, scheduling landing times, predicting weather issues, offering services to the airlines

Air traffic controllers update flight plans

Flight plan update broadcast service

WAN link to other ATC centers

WHERE ARE OBJECTS MOVED OR SHARED?



Flight plan manager tracks current and past flight plan versions

Message bus

. . .

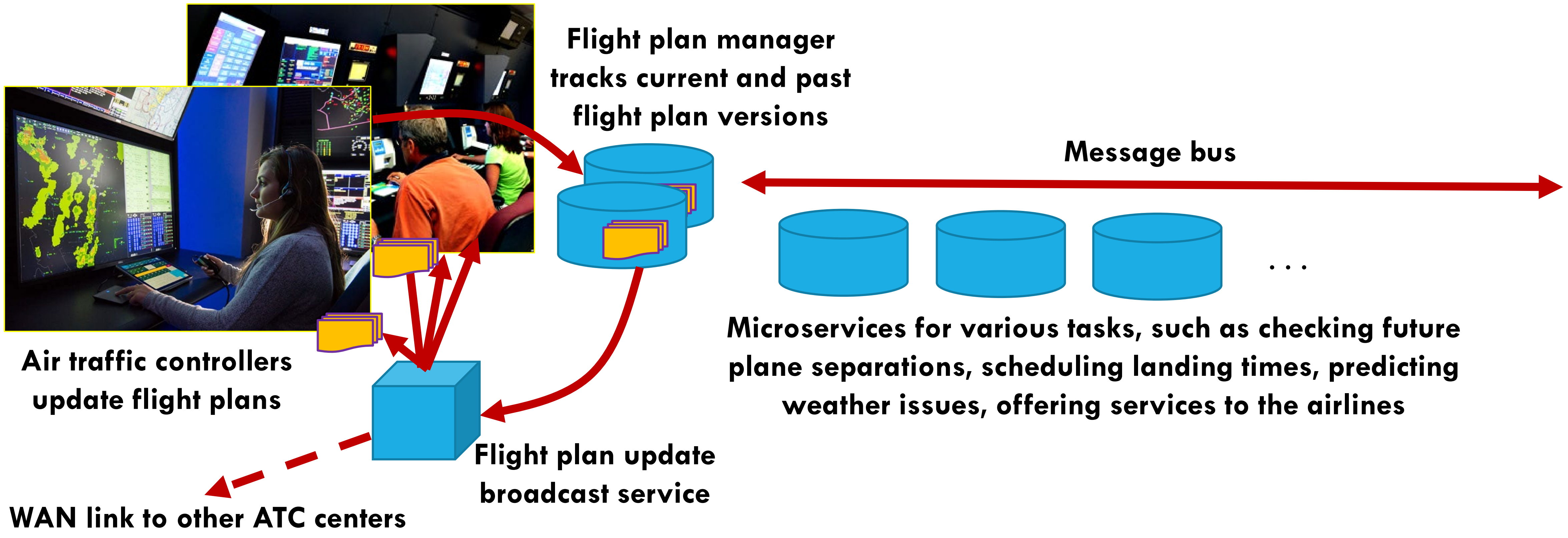
Air traffic controllers update flight plans

Microservices for various tasks, such as checking future plane separations, scheduling landing times, predicting weather issues, offering services to the airlines

Flight plan update broadcast service

WAN link to other ATC centers

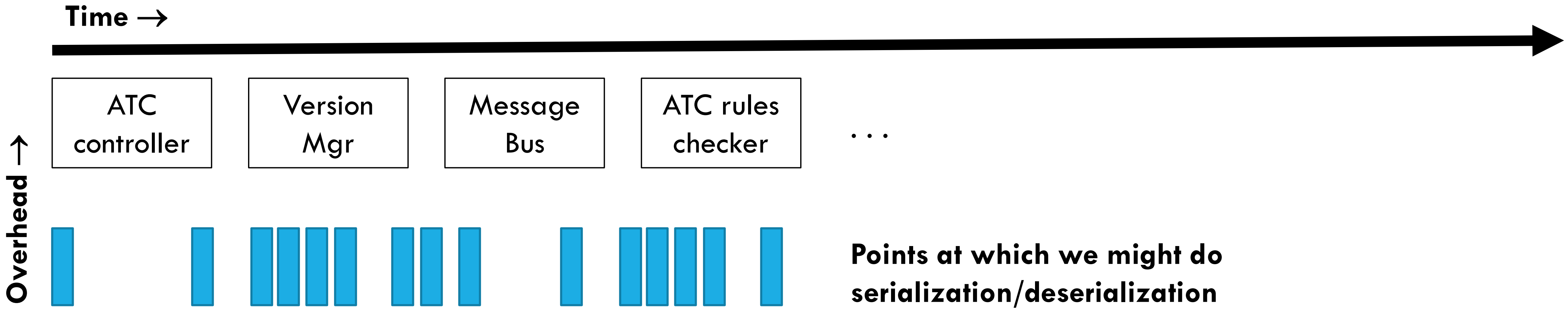
WHERE ARE OBJECTS MOVED OR SHARED?


Flight plan manager tracks current and past flight plan versions
Message bus
. . .
Microservices for various tasks, such as checking future plane separations, scheduling landing times, predicting weather issues, offering services to the airlines
Air traffic controllers update flight plans
Flight plan update broadcast service
WAN link to other ATC centers

WHEN DO WE SERIALIZE/DESERIALIZE?

Each time an object is read or written (from disk or network)

Each time an object is passed from one module to another



COST IMPLICATIONS

Potentially, a major source of overhead!

Often, it is best to store a complex serialized object in a file, and then just pass the file *name* from place to place. Then the CORBA object just has a few bytes in it (very cheap).

In a complex application where the actual fields in the object aren't needed by many modules, this reduces costs dramatically!

WHY WOULD A MODULE NOT LOOK AT THE DATA?

In the air traffic example, some modules just look at a few fields.

The WAN module is responsible for sharing updates with other air traffic control centers. It doesn't need to actually see the details.

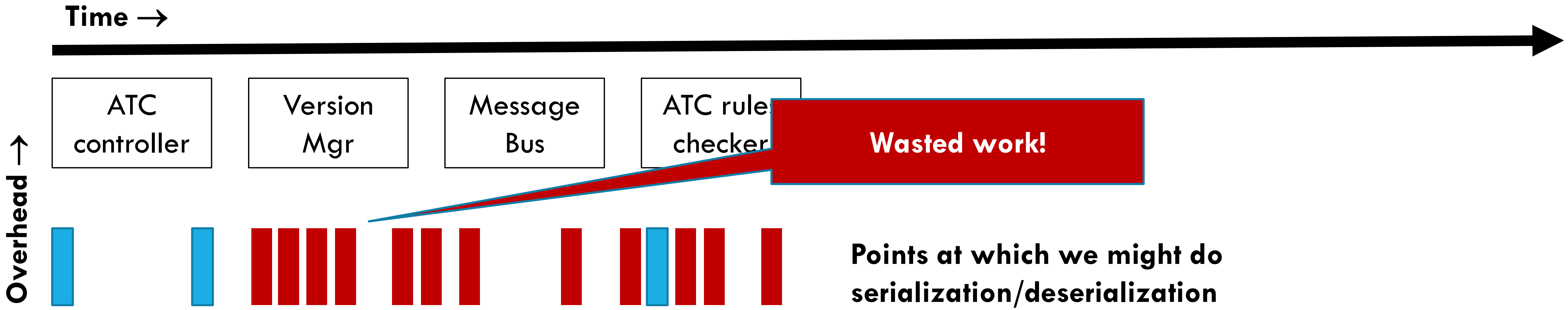
… in fact, several modules simply move objects from process to process.

… all of these would be happy with just sharing the object *name*.

OLD APPROACH

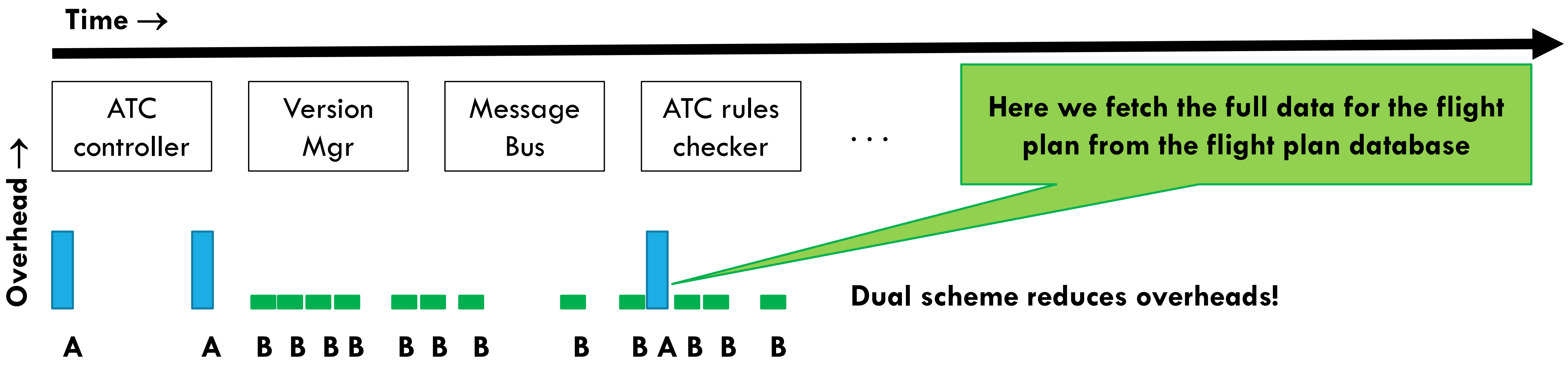
Each time an object is read or written (from disk or network)

Each time an object is passed from one module to another



SHARING OBJECT NAMES, ONLY FETCH THE DATA IF THE MODULE REALLY REQUIRES IT

We only do a costly action when the module will actually touch the inner data fields!



Time →
ATC controller
Version Mgr
Message Bus
ATC rules checker
. . .
Overhead →
Here we fetch the full data for the flight plan from the flight plan database
Dual scheme reduces overheads!
A A B B B B B B B B B A B B B

SUMMARY

Modular design creates a need for processes to share data.

In a single Linux system, pipes and file sharing are by far the most common models. But there are some important uses of shared memory.

The options are easy to use, but we need to be very aware of overheads and costs!