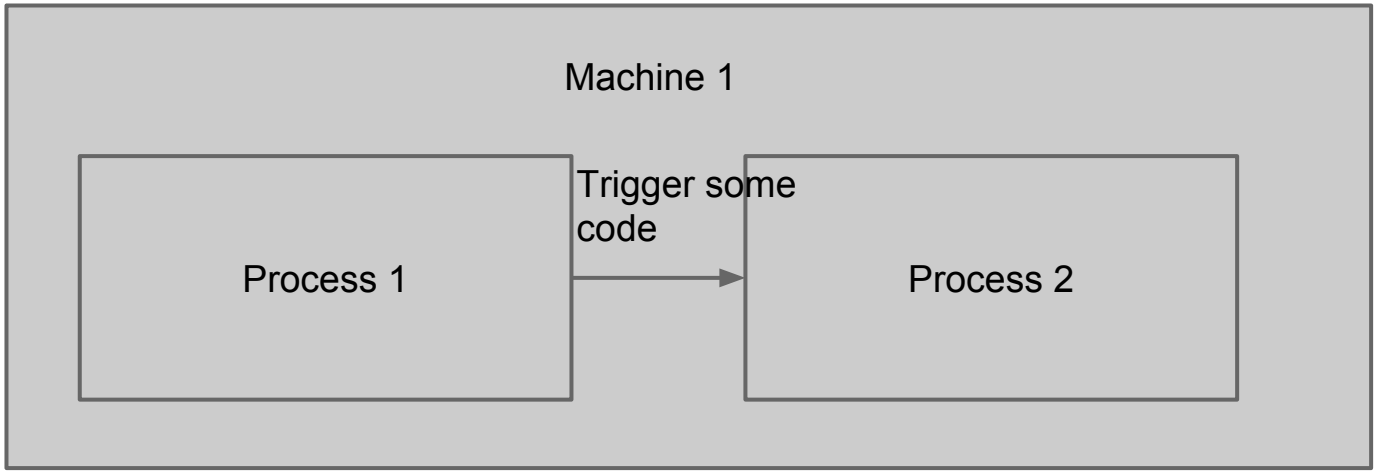
# Local Inter Process Communication

## Local Remote Procedure Call




For .NET developers

# History: Memory

Before: No isolation - all share the same pointers space. Badly written program rewrites operating system or other program memory.

Then: Operation system acts as virtual machine translating virtual points to real pointers. Each program has its own pointers space. Slightly slower, but computers get faster.

# History: Execution

Before: Single execution thread. Infinite loop - executed your code - call next. Windows message pumping (**System.Threading.Dispatcher**) work this way.

Then: Several virtual execution threads. Operating system passes limited number of instructions to CPU of first program. Then runs

# History: Results

1. Multi tasking operating system with process(**System.Diagnostics.Process**) isolation.

2. Operating system provides secure way to share memory(**System.IO.MemoryMappedFiles.MemoryMappedFile**) and synchronize execution of processes ( **System.Threading.Mutex**(...,string **name**), **System.Threading.EventWaitHandle**(..., string **name** ) ).

3. Abstraction are build upon (message passing, procedure call) with different usability and performance characteristics above primitives

# History: Context before RPC

1. Various networking technologies to pass data. No abstractions.
2. C is dominant.

# History: Invention RPC

1. Procedure call look like `right` abstraction for Internet.
2. RPC needs runtime with threading and synchronization.
3. RPC can use different networking protocols.
4. To avoid complex marshalling code copy paste need to define code generation out of some language. C does not supports interfaces description. Invented: Interface Description Language (IDL).
5. Good enough for single machine RPC.

# History: Decline of RPC

1. Fails on internet scale. Binary closed source incompatible implementations are hard to debug and use.
2. Fails on internet scale. There is not easy build in way of doing meta networking (routing, error reporting, failure handling, troubleshooting)
3. Computers gets faster to work with strings. Which are actually easier for start to develop and debug.
4. New programming languages embed IDLs.
5. Windows Communication Foundation(WCF)

# Why local WCF is bad idea in 2014

1. Slow. At least 10-20 times slower than RPC.
2. Big working set. Memory, CPU.
3. Does not support local operation system security models - Integrity Levels.
4. No native counterpart (until Win 8)

# Why RPC is good/bad idea in 2014

Good:

1. RPC is build in into OS - rpcrt4.dll
2. Fast.

Bad:

1. No full compatibility with .NET.
2. No versatile callbacks, timeouts, asynchrony build in.

# What is good idea

Develop something similar to WCF, RPC, COM but with desired modifications.

Similarity is needed to have already documented and known stuff for developers.