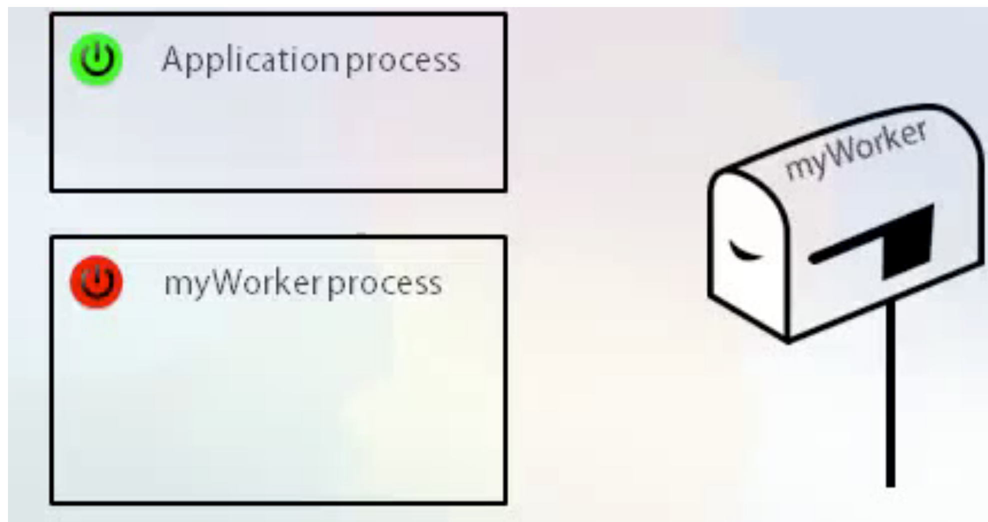
Nothing worth saying
fits on a slide…

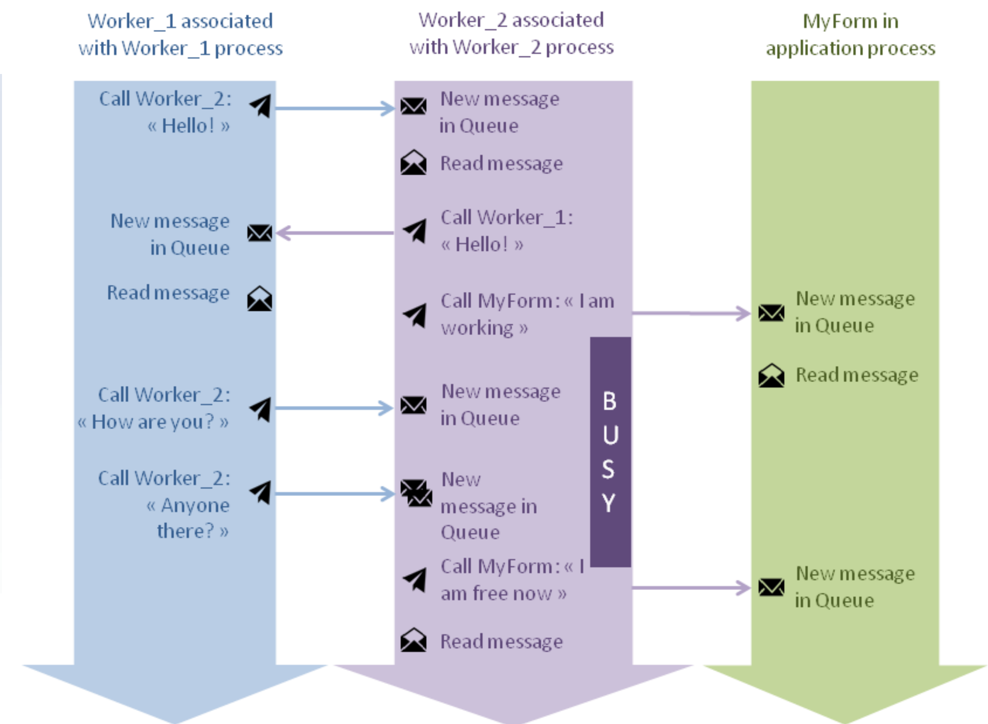
CALL FORM

- This is a superior alternative to **CALL PROCESS**/On Outside Call
- There is no form event associated with **CALL FORM**
- Special rules for form startup
- Calls are appended to the current thread of execution

Understanding the commands

- The basic command documentation is excellent
- The “About workers” documentation is confusing and misleading





Understanding the execution sequence

* Requests from one source
* Requests from multiple sources
* Self-calls
* Form loading sequences

Code in run order: What order do the alerts appear in?

```
  // TryPhrases process controller
C_LONGINT(TryPhases_winref)
TryPhases_winref:=Open form window("TryPhases";Plain form window)
CALL FORM(TryPhases_winref;"ShowAlert";"1: Before displaying form.")
DIALOG(“TryPhases")
```

```
  // Object method for a button on the form
CALL FORM(TryPhases_winref;"ShowAlert";"2: Set in On Load of object
method")
```

```
 // Form Method: TryPhases
Case of
  : (Form event=On Load)
    CALL FORM(TryPhases_winref;"ShowAlert";"3: Set in On Load.")
    ShowAlert ("4: Called in On Load")
End case
```

Code in run order: What order do the alerts appear in?

Code

```
  // TryPhrases process controller
C_LONGINT(TryPhases_winref)
TryPhases_winref:=Open form window("TryPhases";Plain form window)
```

Window 3454555 Queue

[ ]

Code in run order: What order do the alerts appear in?

Code

```
  // TryPhrases process controller
C_LONGINT(TryPhases_winref)
TryPhases_winref:=Open form window("TryPhases";Plain form window)

CALL FORM(TryPhases_winref;"ShowAlert";"1: Before displaying form.")
```

Window 3454555 Queue

```
[]

["ShowAlert";"1: Before displaying form."]
```

Code in run order: What order do the alerts appear in?

Code

```
  // TryPhrases process controller
C_LONGINT(TryPhases_winref)
TryPhases_winref:=Open form window("TryPhases";Plain form window)
```

```
CALL FORM(TryPhases_winref;"ShowAlert";"1: Before displaying form.")

DIALOG(“TryPhases")
```

```
  // Object method for a button on the form
CALL FORM(TryPhases_winref;"ShowAlert";"2: Set in On Load of object method")
```

Window 3454555 Queue

```
[]
```

```
["ShowAlert";"1: Before displaying form."]
```

```
["ShowAlert";"1: Before displaying form."]
["ShowAlert";"2: Set in On Load of object method"]
```

Code in run order: What order do the alerts appear in?

Code

Window 3454555 Queue

```
 // TryPhrases process controller
C_LONGINT(TryPhases_winref)
TryPhases_winref:=Open form window("TryPhases";Plain form window)
```

→ []

```
CALL FORM(TryPhases_winref;"ShowAlert";"1: Before displaying form.")
```

→ ["ShowAlert";"1: Before displaying form."]

```
DIALOG("TryPhases")
```

```
 // Object method for a button on the form
CALL FORM(TryPhases_winref;"ShowAlert";"2: Set in On Load of object method")
```

→
["ShowAlert";"1: Before displaying form."]
["ShowAlert";"2: Set in On Load of object method"]

```
 // Form Method: TryPhases
Case of
  : (Form event=On Load)

     CALL FORM(TryPhases_winref;"ShowAlert";"3: Set in On Load.")

     ShowAlert ("4: Called in On Load")
End case
```

→
["ShowAlert";"1: Before displaying form."]
["ShowAlert";"2: Set in On Load of object method"]
["ShowAlert";"3: Set in On Load."]

Code in *execution* order

```
  // TryPhrases process controller
C_LONGINT(TryPhases_winref)
TryPhases_winref:=Open form window("TryPhases";Plain form window)
DIALOG("TryPhases")

  // Object method for a button on the form

 // Form Method: TryPhases
Case of
  : (Form event=On Load)
    ShowAlert ("4: Called in On Load")
End case

ShowAlert("1: Before displaying form.")
ShowAlert("2: Set in On Load of object method.")
ShowAlert("3: Set in On Load.")
```

Understanding the execution sequence

* Requests from one source
* Requests from multiple sources
* Self-calls
* Form loading sequences

Advice on structuring form code

* Make the sequence in the code the same as the sequence of execution as much as possible
* Consolidate form setup code, form methods, and scripts into global methods with action/switch/event parameters parameters
* Tip: It's easier to reuse and test form control code when you can emulate form events with a parameter

CALL WORKER

* Designed as a thread-safe way to communicate with preemptive workers
* Useful with any worker in 32 or 64-bit, compiled or interpreted, cooperative or preemptive
* Calls are appended to the current thread of execution

Code execution commands

Command	Location Identifier	Method	Result	{Parameters}	{Parameters}
EXECUTE METHOD	Current context	Name	Result	Parameter 1	Parameter *n*
EXECUTE METHOD IN SUBFORM	Subform reference	Name	Result	Parameter 1	Parameter *n*
EXECUTE ON CLIENT	Client reference	Name	n/a	Parameter 1	Parameter *n*
Execute on server	Process name	Name	Process ID	Parameter 1	Parameter *n*
New process	Process name	Name	Process ID	Parameter 1	Parameter *n*
CALL FORM	Window reference	Name	n/a	Parameter 1	Parameter *n*
CALL WORKER	Worker reference	Name	n/a	Parameter 1	Parameter *n*

Code Execution Commands: Renamed

Command	Location Identifier	Method	Result	{Parameters}	{Parameters}
EXECUTE METHOD	Current context	Name	Result	Parameter 1	Parameter *n*
EXECUTE METHOD IN SUBFORM	Subform reference	Name	Result	Parameter 1	Parameter *n*
EXECUTE ON CLIENT	Client reference	Name	n/a	Parameter 1	Parameter *n*
Execute on server	Process name	Name	Process ID	Parameter 1	Parameter *n*
New process	Process name	Name	Process ID	Parameter 1	Parameter *n*
EXECUTE METHOD IN WINDOW	Window reference	Name	n/a	Parameter 1	Parameter *n*
EXECUTE METHOD IN WORKER	Worker reference	Name	n/a	Parameter 1	Parameter *n*

Code Execution Commands: Renamed (2)

Command	Location Identifier	Method	Result	{Parameters}	{Parameters}
EXECUTE METHOD	Current context	Name	Result	Parameter 1	Parameter *n*
EXECUTE METHOD IN SUBFORM	Subform reference	Name	Result	Parameter 1	Parameter *n*
EXECUTE METHOD ON CLIENT	Client reference	Name	n/a	Parameter 1	Parameter *n*
Execute method on server	Process name	Name	Process ID	Parameter 1	Parameter *n*
Execute method in new process	Process name	Name	Process ID	Parameter 1	Parameter *n*
EXECUTE METHOD IN WINDOW	Window reference	Name	n/a	Parameter 1	Parameter *n*
EXECUTE METHOD IN WORKER	Worker reference	Name	n/a	Parameter 1	Parameter *n*

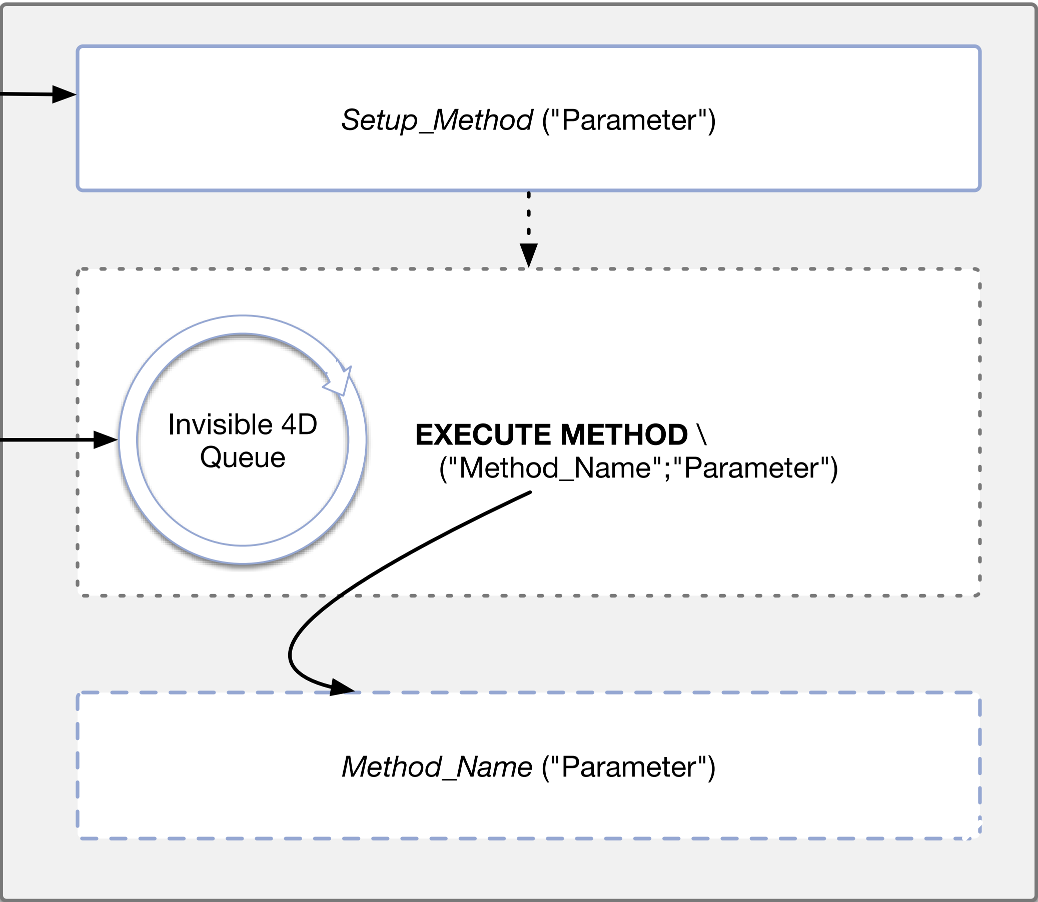
Execution context in more detail

* Thread of execution
* Process state/context

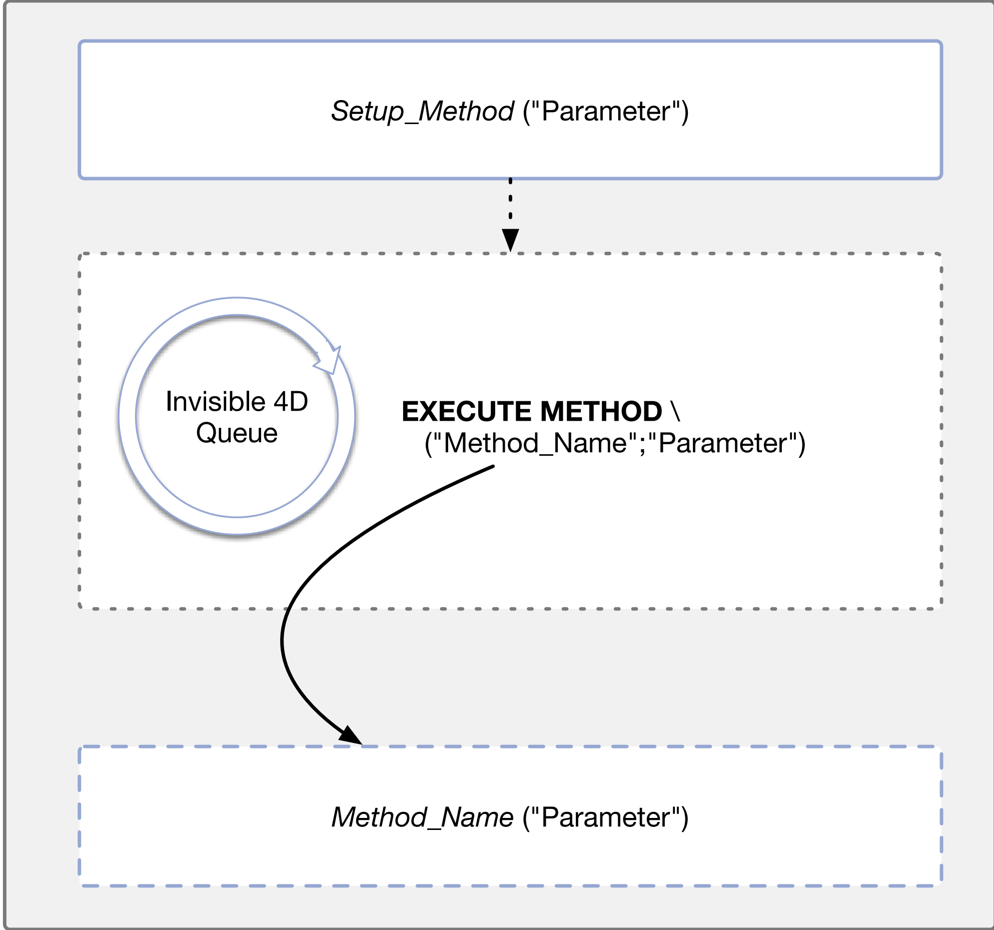
Worker Thread of Control

CALL WORKER \
("Worker";"Setup_Method";"Parameter")

CALL WORKER \
("Worker";"Method_Name";"Parameter")



Worker Thread of Control



Worker Context

- Current records
- Locked records
- Read-write states
- Current selections
- Process named selections
- Interprocess named selections
- Process sets
- Interprocess sets
- Transactions
- Process variables
- Process arrays
- Open documents

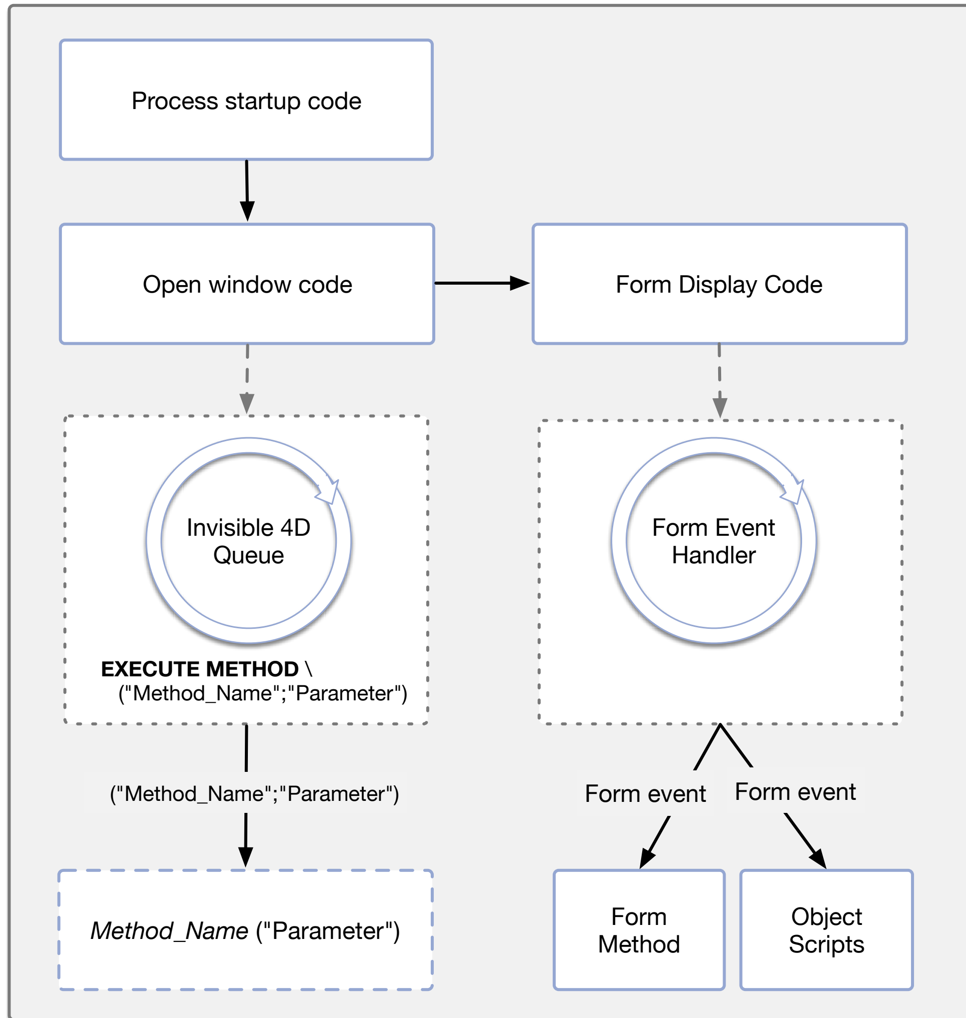
Advice on structuring worker code

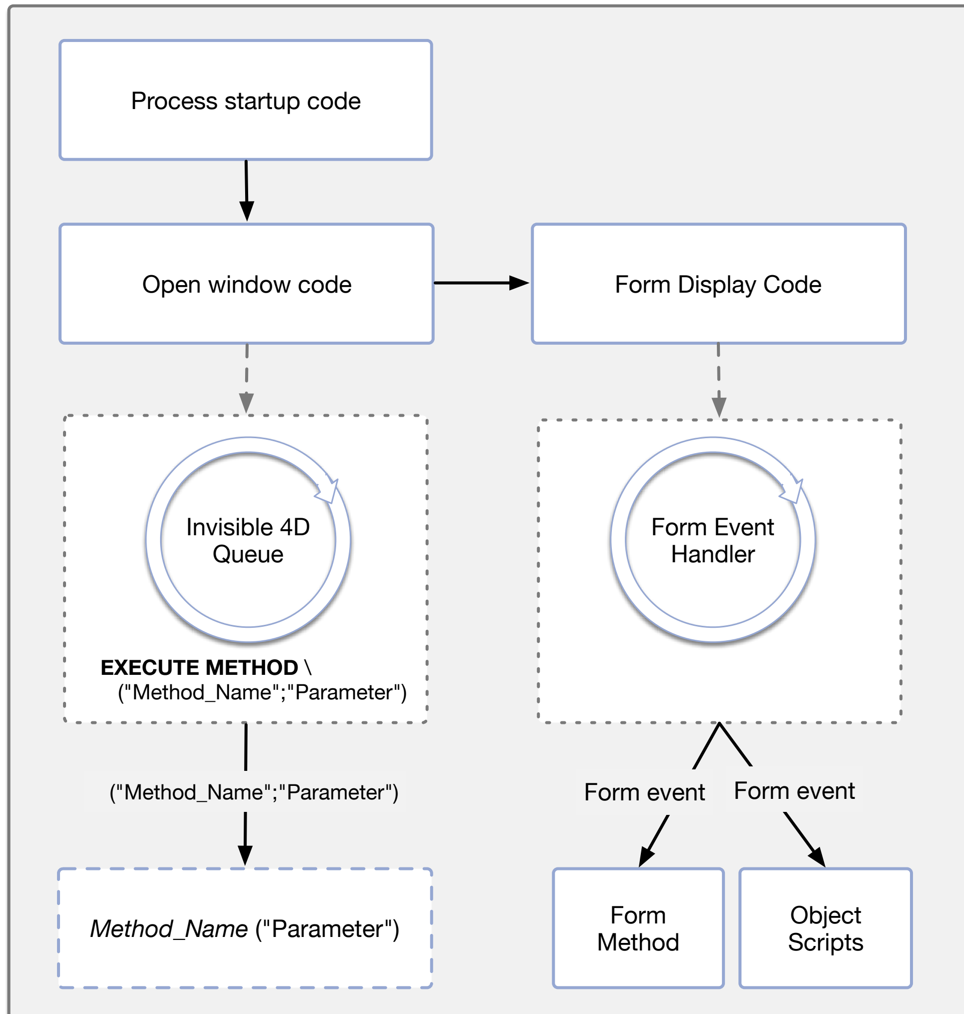
- Keep the code as “straight-line” as possible
- Pass **C_OBJECT** messages
- Think carefully before making self-calls in a worker
- Tip: You can often identify self-calls by checking the process name and origin

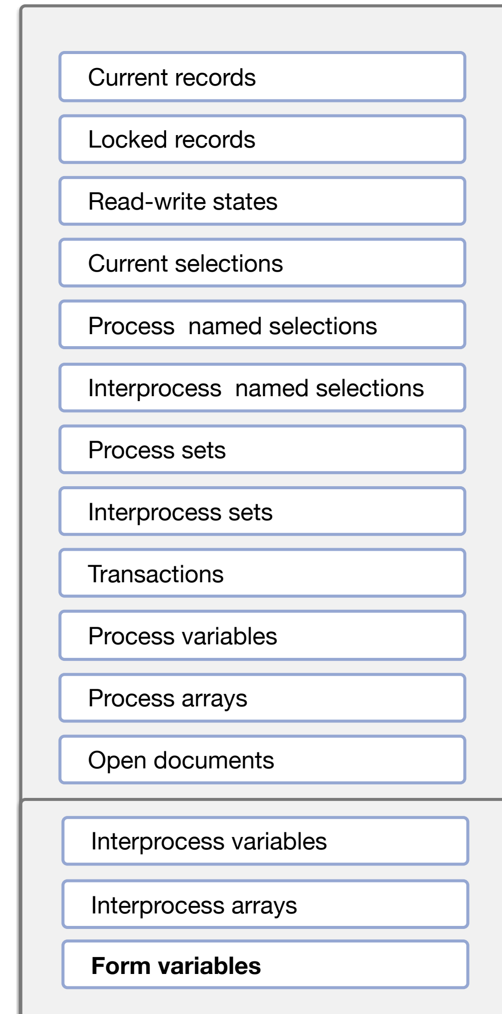
Food for thought: Recursion

```
  // Recurse
If (Undefined(Recurse_count))
  Recurse_count:=0
End if
Recurse_count:=Recurse_count+1

Recurse


// Recurse_Worker
If (Undefined(Recurse_count))
  Recurse_count:=0
End if
Recurse_count:=Recurse_count+1

CALL WORKER(“Recurse_Worker";"Recurse_Worker")
```

Form Thread of Control




Form Thread of Control
Process startup code
Open window code
Form Display Code
Invisible 4D Queue
Form Event Handler
EXECUTE METHOD \
("Method_Name";"Parameter")
("Method_Name";"Parameter")
Form event
Form event
Method_Name ("Parameter")
Form Method
Object Scripts


Window Context
Current records
Locked records
Read-write states
Current selections
Process named selections
Interprocess named selections
Process sets
Interprocess sets
Transactions
Process variables
Process arrays
Open documents
Interprocess variables
Interprocess arrays
Form variables

4D WORLD TOUR

Multi-window Process

1° Click on "Open MyForm"
2° Enter a message.
3° Click on "Call Form" button. You can repeat this step with differents messages.
4° Click on "Close MyForm"

MyForm

Open MyForm

Message: Hello!

Call MyForm

Close MyForm

MyForm2

Open MyForm2

Message: Hi!

Call MyForm2

Close MyForm2

MyForm

Clear log

Log

MyForm2

Clear log

Log

Form Context

Form variables

Form variables

Form variables

Process Context

- Current records
- Locked records
- Read-write states
- Current selections
- Process named selections
- Interprocess named selections
- Process sets
- Interprocess sets
- Transactions
- Process variables
- Process arrays
- Open documents
- Interprocess variables
- Interprocess arrays

Advice on structuring form code

* Make the sequence in the code the same as the sequence of execution as much as possible
* Consolidate form setup code, form methods, and scripts into global methods with action/switch/event parameters parameters
* Tip: It's easier to reuse and test form control code when you can emulate form events with a parameter

Review: Messages are code

- “Messages” are **blocks of code**
- `eval()` is **inherently dangerous** in any language.
  Use with care.
- Code executes in the **context of the target**

Review: Queues

- Calls go into a **form** (window) **queue** or a **worker queue**
- Calls send **messages** if you're 4D, they send **remote procedure calls** if you're a 4D developer.
- Queues cannot be inspected, counted, serialized, sorted, or seen in any way

Review: Call sequence

* Calls are always sent and received **in sequence**
* Calls from multiple sources are received in an **unpredictable** order relative to one another
* Delivery is **guaranteed** unless the queue is destroyed

Review: Features

- There is **no** automatic response system
- Calls are always sent and received **in sequence**
- Only windows and workers can **receive** calls
- Any piece of code can **send** messages
- The queues are internal and invisible
- Incoming code cannot be reviewed or blocked

Review: Forms

* There is no form event triggered by **CALL FORM**
* The form method is not triggered by **CALL FORM**
* A method invoked by **CALL FORM** has access to all of the variables, etc. of the target process and to the form variables of the specified window’s form
* Incoming calls cannot be blocked

Review: Workers

- Workers are available in all flavors of 4D, including 32 and 64-bit, compiled and interpreted, 4D, 4D Remote and 4D Server
- Workers cannot block incoming calls
- Workers cannot inspect or review incoming calls
- Workers are started or restarted automatically
- **KILL WORKER** destroys a worker and may destroy any number of pending messages

Publish-Subscribe: Demonstration

Clients

First | Previous | Next | Last | Delete | Cancel | Validate

Client

First_Name : Charo

Last_Name : Atcock

Email : cunderhill3@marriott.com

UUID_PK : 633617554F874D5592B8BB82CE68BC8D

Daytime_Phone : 62-(238)535-9066

Nighttime_Phone : 1-(336)658-5356

Address : 43221 Rockefeller Center

City : Bajomulyo

Visits for Sadie

Date	Age at Visit (Days)	Procedure
Jun 3, 2017	10	vaccination/worm
Jun 8, 2017	15	stuff
Jun 15, 2017	22	anal glands

3 visits found

Pets

Name	Birthday	Species
Sadie	May 24, 2017	Dog
Sam	Nov 27, 2008	Snake

2 pets found

Dogs : Practice and Birth Year

Pet Type Relative to Practice and Birth

Publish-Subscribe: Form sections

Client

text	text
text	text
text	text
text	text

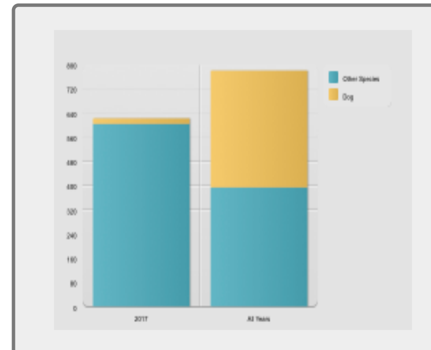
VisitList

Heading	Heading	Heading	Heading
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell

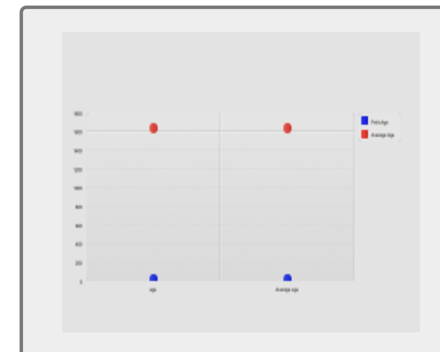
PetList

Heading	Heading	Heading
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell

PetGraph



ProcedureGraph



4D WORLD TOUR

Publish-Subscribe: On Set Pet

Client

text	text
text	text
text	text
text	text

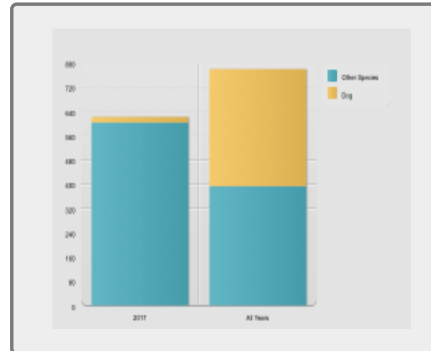
VisitList

Heading	Heading	Heading	Heading
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell

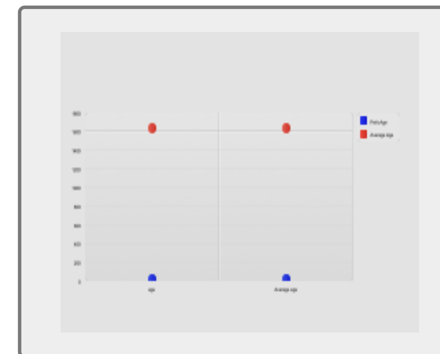
PetList

Heading	Heading	Heading
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell

PetGraph



ProcedureGraph



4D WORLD TOUR

Publish-Subscribe: On Set Visit

Client

text	text
text	text
text	text
text	text

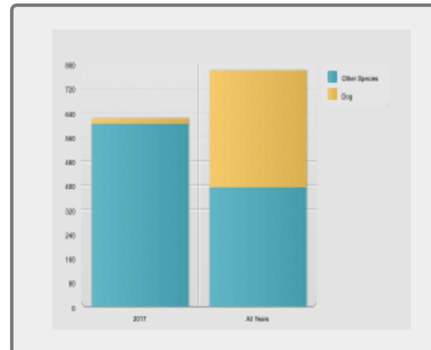
VisitList

Heading	Heading	Heading	Heading
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell

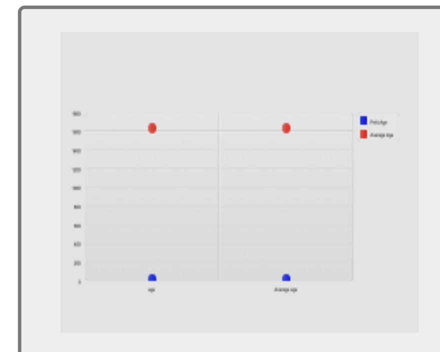
PetList

Heading	Heading	Heading
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell

PetGraph



ProcedureGraph



Publish-Subscribe: On Set Pet: Message-based

MessageHub

Client

text	text
text	text
text	text
text	text

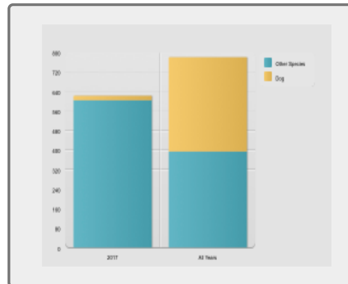
VisitList

Heading	Heading	Heading	Heading
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell

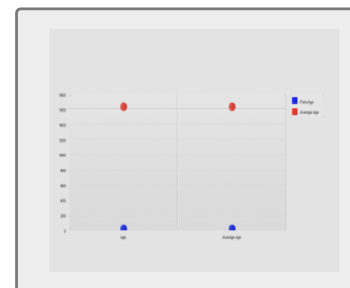
PetList

Heading	Heading	Heading
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell

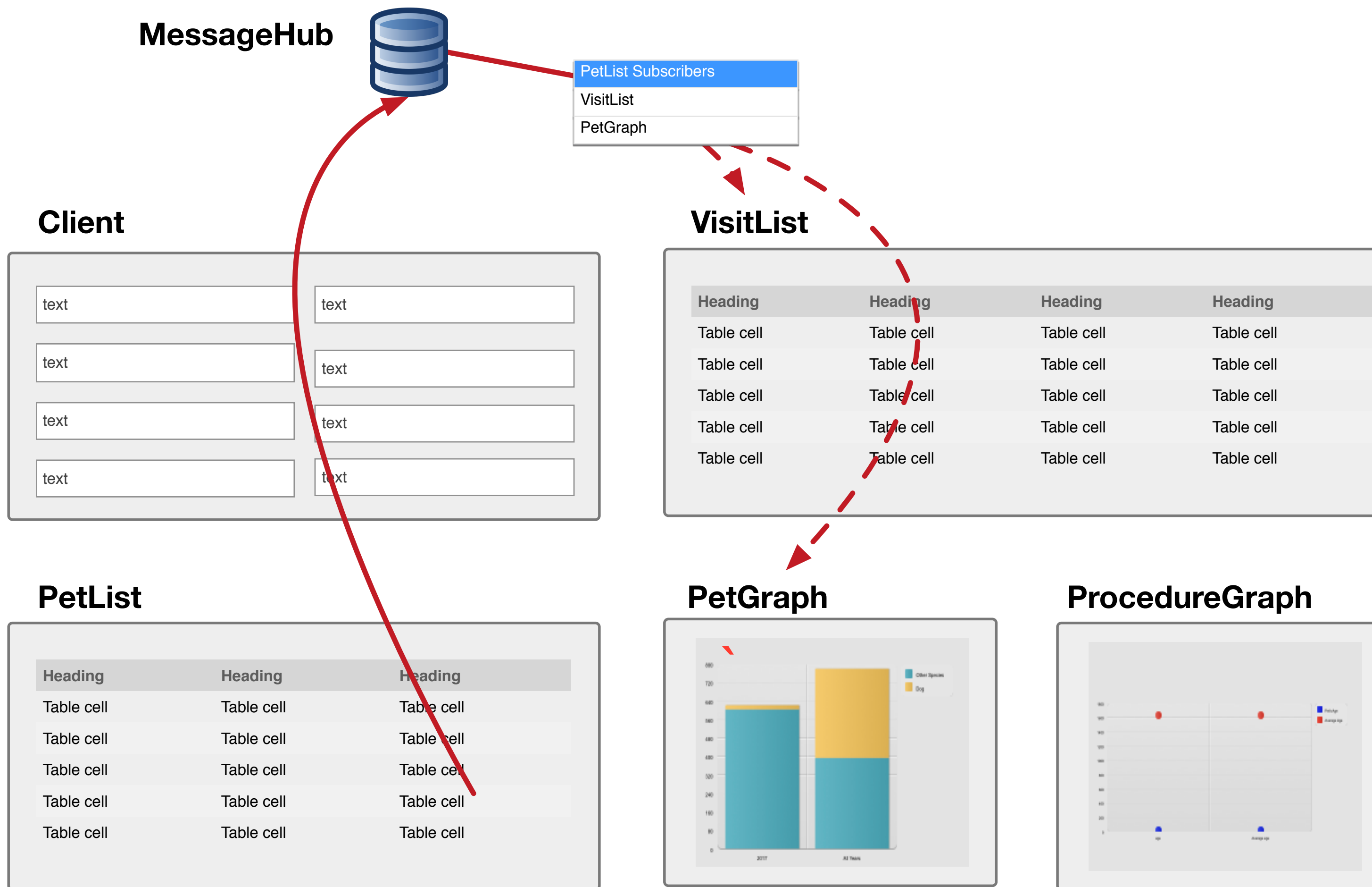
PetGraph



ProcedureGraph



Publish-Subscribe: On Set Pet: Message distribution



Publish-Subscribe: On Set Visit: Message-based

MessageHub

Client

text	text
text	text
text	text
text	text

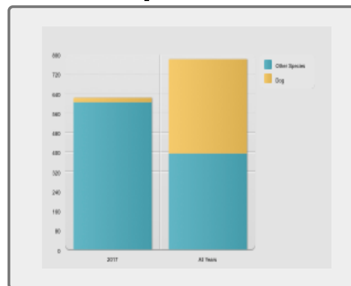
VisitList

Heading	Heading	Heading	Heading
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell
Table cell	Table cell	Table cell	Table cell

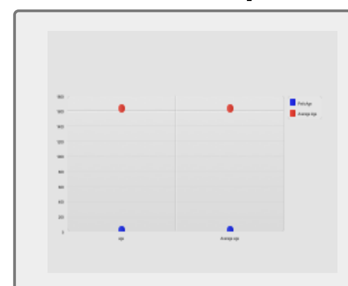
PetList

Heading	Heading	Heading
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell
Table cell	Table cell	Table cell

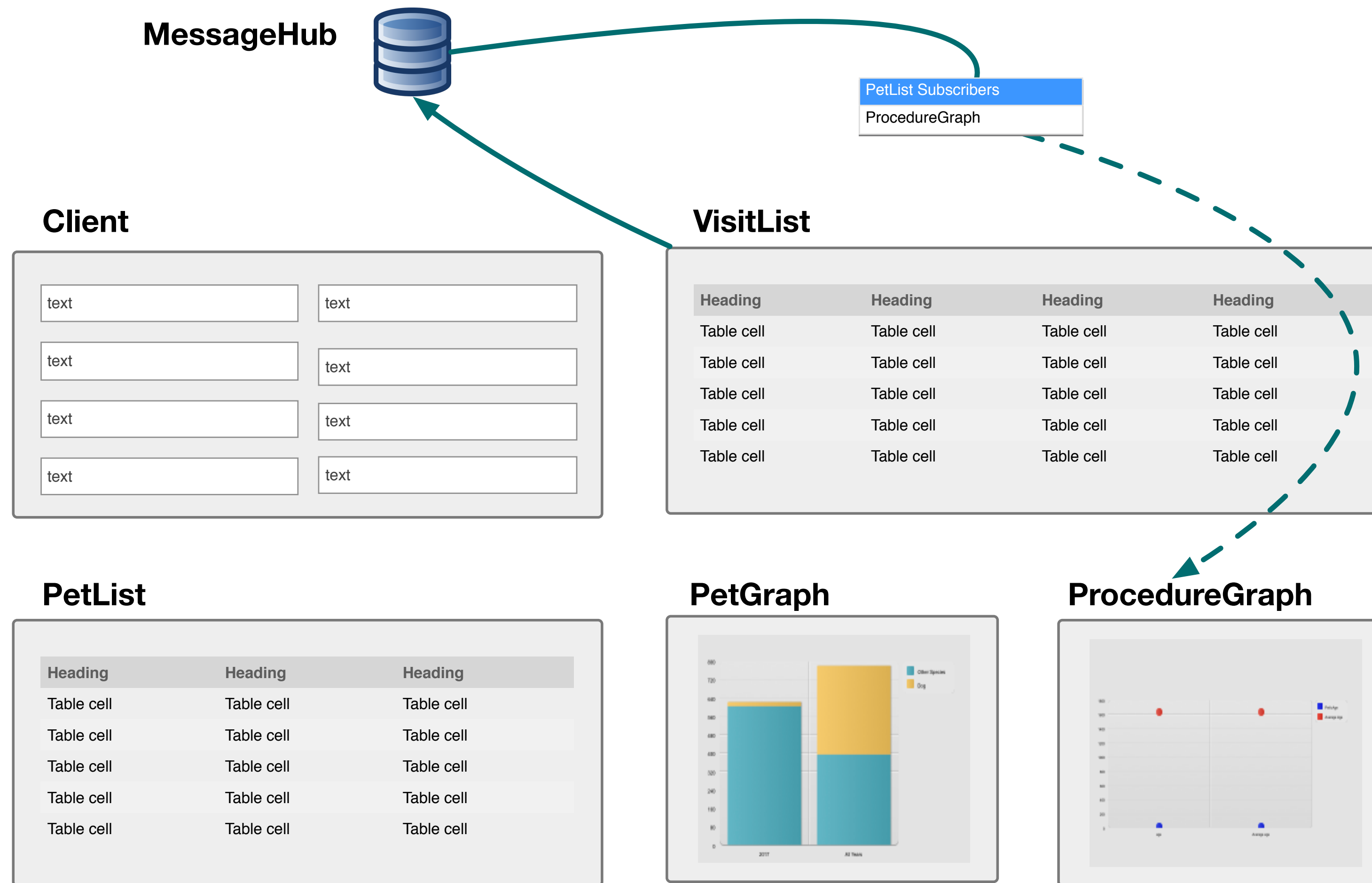
PetGraph



ProcedureGraph



Publish-Subscribe: On Set Visit: Message distribution

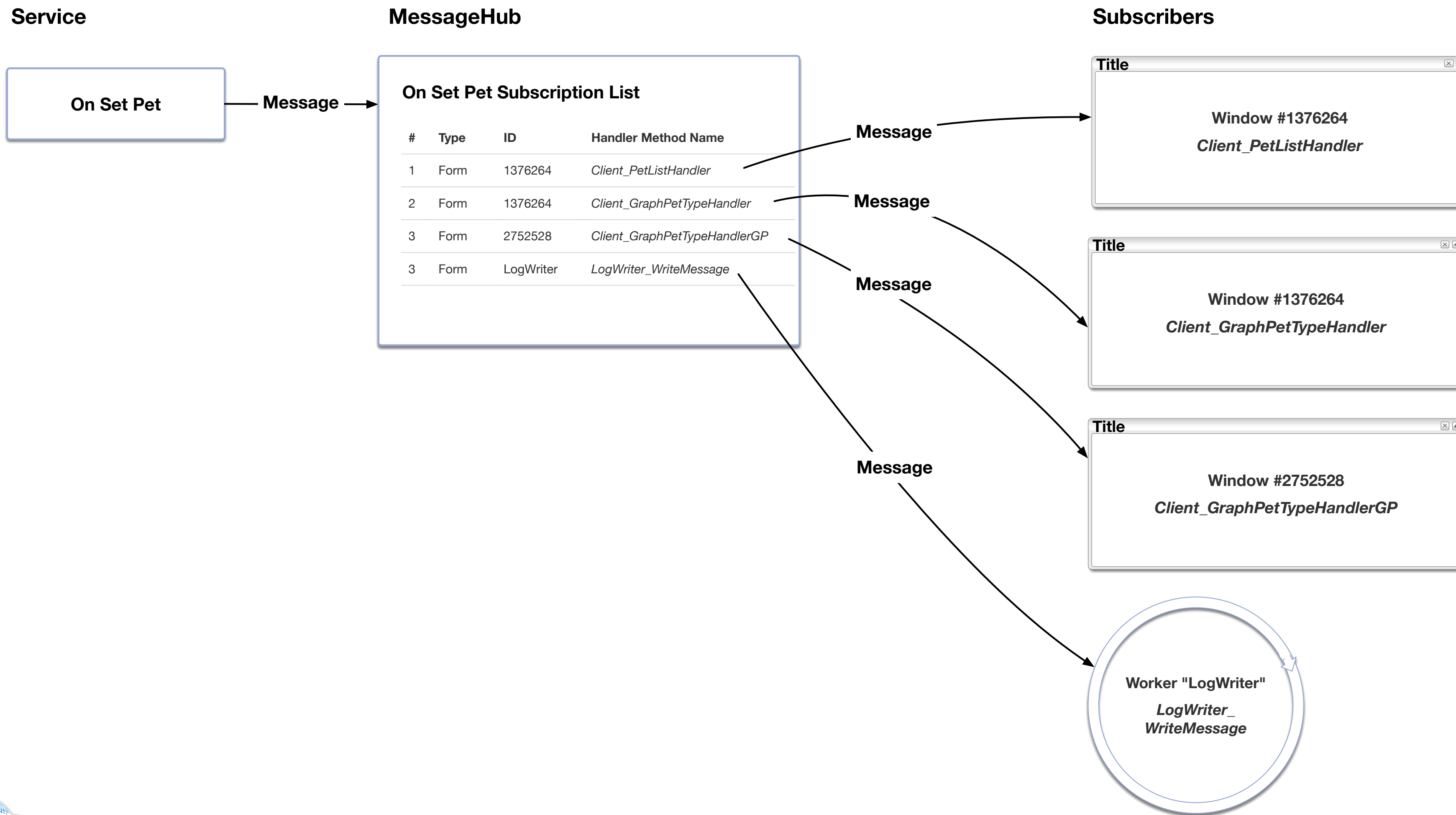


4D WORLD TOUR

Subscription list


```
{
   "OnSetPet":{
      "host":"MacBook Pro",
      "name":"VetClient_Service_OnSetPet",
      "service_type":"Service_Is_Subscription",
      "publisher_type":"Service_publisher_is_a_form",
      "publisher_id":"1376264",
      "publisher_handler":"Client_PetListHandler",
      "publiserh_winref":1376264,
      "subscriptions_count":3,
      "subscriptions":[
         {
            "recipient":{
               "host_name":"MacBook Pro",
               "method_name":"Client_GraphPetTypeHandler",
               "type":"Form",
               "tag":"VetClient_Service_OnSetPet",
               "winref":1376264
            }
         },
         {
            "recipient":{
               "host_name":"MacBook Pro",
               "method_name":"Client_PetListHandler",
               "type":"Form",
               "tag":"VetClient_Service_OnSetPet",
               "winref":1376264
            }
         },
         {
            "recipient":{
               "host_name":"MacBook Pro",
               "method_name":"Client_VisitListHandler",
               "type":"Form",
               "tag":"VetClient_Service_OnSetPet",
               "winref":1376264
            }
         }
      ]
   }
}
```

Publish-Subscribe: Sample distribution



4D WORLD TOUR

Keep coupling loose

Coupling describes how tightly a class or routine is related to other classes or routines. The goal is to create classes and routines with small, direct, visible, and flexible relations to other classes and routines, which is known as "loose coupling."....

Good coupling between modules is loose enough that one module can easily be used by other modules. Model railroad cars are coupled by opposing hooks that latch when pushed together. Connecting two cars is easy—you just push the cars together. Imagine how much more difficult it would be if you had to screw things together, or connect a set of wires, or if you could connect only certain kinds of cars to certain other kinds of cars. The coupling of model railroad cars works because it's as simple as possible. In software, make the connections among modules as simple as possible.

*— Steve McConnell, **Code Complete**, 2nd Edition*

Publish-Subscribe: Sample message

```
{
   "message_header": {
      "service_host": "MacBook Pro",
      "service_name": "VetClient_Service_OnSetPet",
      "sender_host": "MacBook Pro",
      "sender_name": "Client_Pet_Listbox",
      "message_type": "Event",
      "event": "VetClient_Event_OnSetPet",
      "tag": "VetClient_Service_OnSetPet"
   },
   "message_payload": {
      "pet_uuid": "720DDEA5642E474CB6B15B9361DF78BE"
   }
}
```

Coupling and calls

- The publisher should not know about the **internals** of its subscribers
- The publisher should not know about the **state** of its subscribers
- The publisher should not know about the **existence** of its subscribers

Designing Calls: Message types: Signal


```
{
    "event":"pet_selected"
}
```

Designing Calls: Message types: Identifier

```
{
    "event":"pet_selected",
    "pet_uuid":"720DDEA5642E474CB6B15B9361DF78BE"
}
```

Designing Calls: Message types: Full payload


```
{
   "event":"pet_selected",
   "pet":{
      "UUID_PK":"720DDEA5642E474CB6B15B9361DF78BE",
      "Name":"Sam",
      "Client_UUID":"633617554F874D5592B8BB82CE68BC8D",
      "Date_of_Birth":"2008-11-26T13:00:00.000Z",
      "Species_UUID":"D59F89B3BE0C4E34989B73BED0E7B68B"
   }
}
```

Designing Calls: Message types: Command (bad)

```
{
    "event":"pet_selected",
    "pet_uuid":"720DDEA5642E474CB6B15B9361DF78BE",
    "command":"reload_related_visits"
}
```

Be polite: **Say** what happened, don’t **tell** what to do

Yes :)

- Row selected (ID)
- Row deleted (ID)
- Row updated (ID)
- Query run (Conditions)

No :(

- Highlight button
- Set font color to red
- Load related records
- Cancel record

Coupling and message formats

* Do you have to update the subscribers when the publisher changes the message payload?
* Do you have to pass a lot of data useful only to one or a few types of subscriber?
* Are payloads “heavy” and hard to process or maintain?

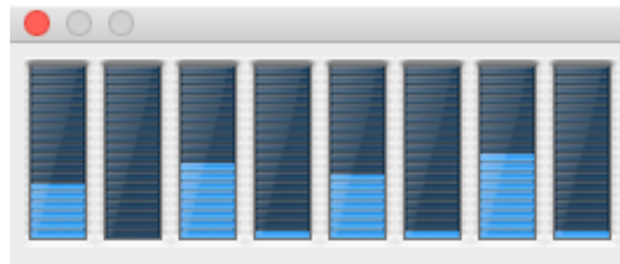
Designing Calls: Be parsimonious

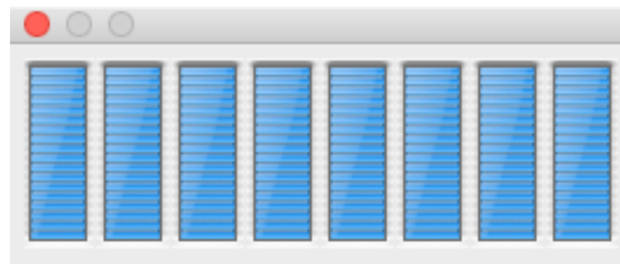
* In 4D, you can **take the database for granted**. In other environments, you often have to package details into massive objects. It's not great to work with.
* How much does a service need to know about its subscribers? Ideally, **nothing at all**. It doesn't even need to know if there are subscribers.
* It's tempting to pack a lot of data into the message payload. Resist. **Send as little as practical and efficient**. Use the database.

Message format tips

* Consider an overall message envelope with a **`header`** and **`body`**
* Header meta-data like `custom_type` and `version` can really save you down the line.
* Headers are useful for any sort of messaging system that wants to support broadcasting or routing
* Keep messages as simple as you can
* Make tools to capture and validate messages
* Keep message formats as similar to each other as practical

Background: Cooperative versus Preemptive Processing

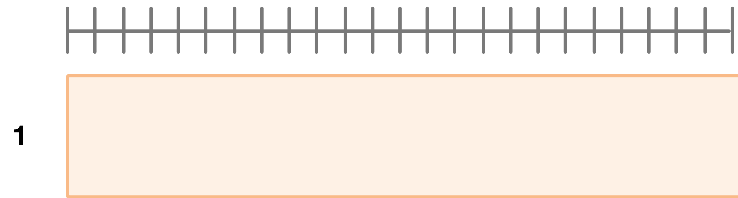


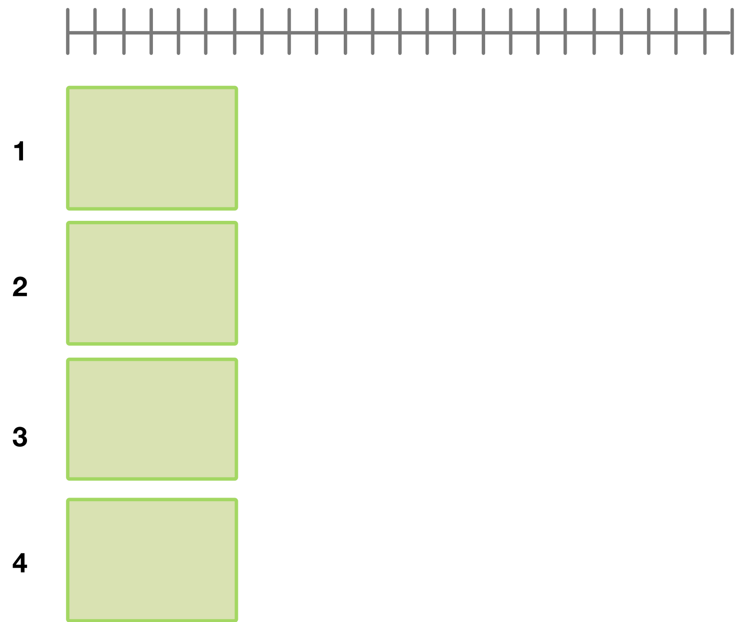


Preemptive Processing Requirements

* 64-bit
* Compiled
* Preemptive flag on
* 4D or 4D Server
* **CALL WORKER**, **New process**, or **Execute on server**
* Thread-safe commands

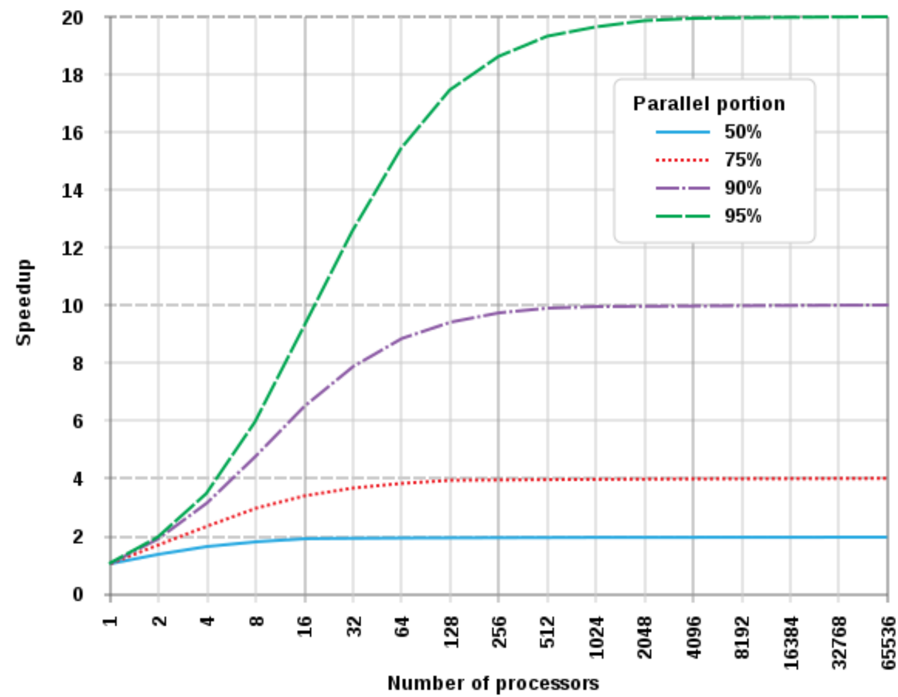
Benefits of preemptive processing: Idealized view





Amdahl’s Law

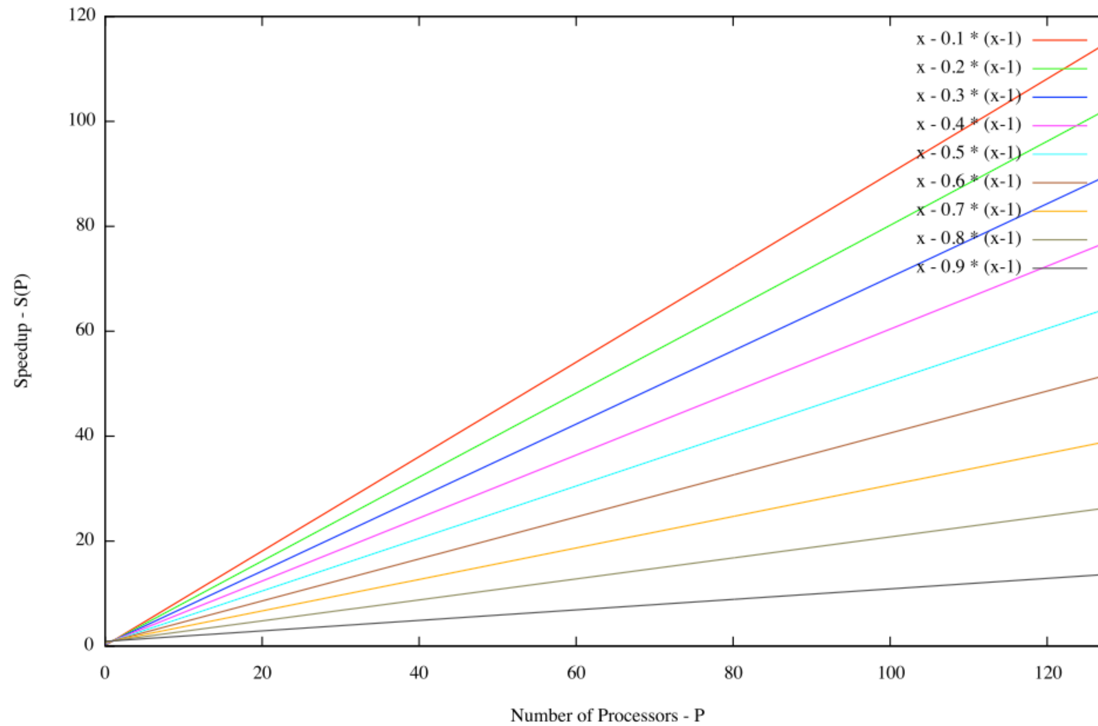
$$S_{\text{latency}}(s) = \frac{1}{(1 - p) + P/s}$$



https://en.wikipedia.org/wiki/Amdahl%27s_law

Gustafson's law

$$S_{\text{latency}}(s) = 1 \quad p + sp$$



https://en.wikipedia.org/wiki/Gustafson%27s_law

Amdahl's Law

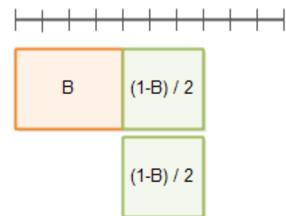
The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program

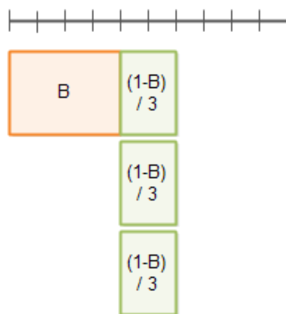
The more cores you add to a CPU, the faster the parallel parts of an application are processed, so the more the performance becomes dependent on the performance in the sequential parts

Multi-core and multi-threading performance (the multi-core myth?)

https://scalibq.wordpress.com/2012/06/01/multi-core-and-multi-threading/

Amdahl's Law: Real-world performance





http://tutorials.jenkov.com/java-concurrency/amdahls-law.html

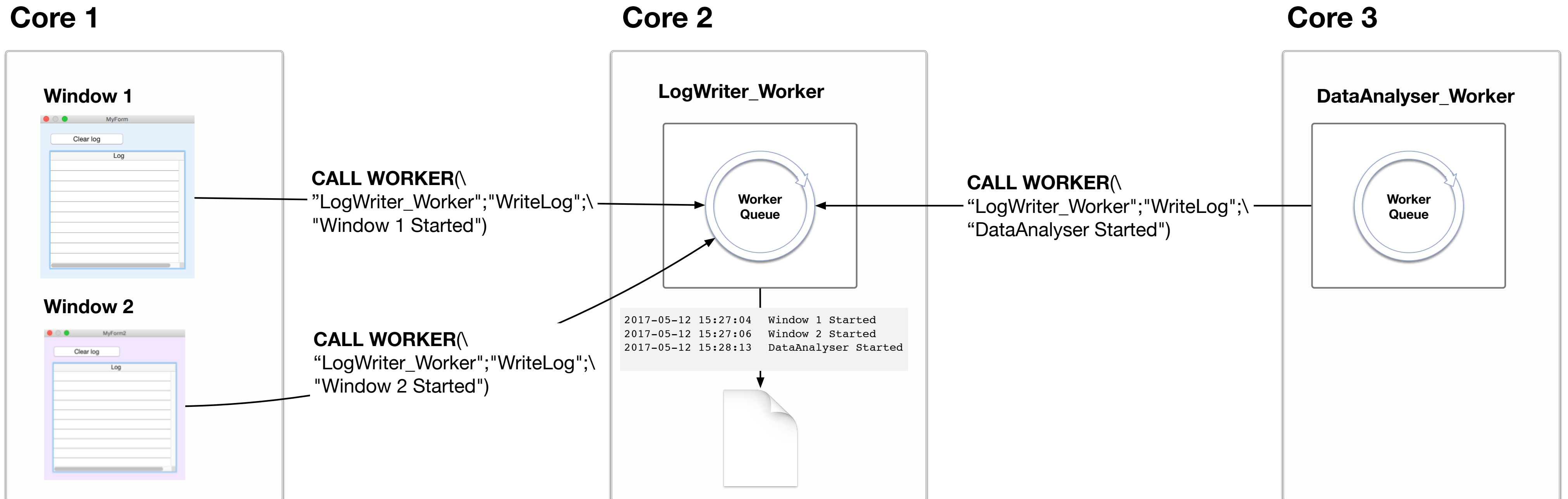
Choosing tasks to run on secondary cores

- Only some tasks are suitable for optimization using multiple threads
- Chose **discrete**, **time-consuming**, **CPU-intensive** tasks
- Time spent on **preparing** work and **marshaling** results counts against the benefit of extra CPUs

Decent Candidates for Preemptive Processing

* Centralized log writing
* Folder-watch system for imports
* Background summarization for fact tables, reports, or other extracts and analysis
* Other ideas?

Conceptual Example: LogWriter



Other Ways to Get More Speed

* 4D Compiler
* SSD
* Idle processes
* RAIC
* Network-based systems with an API
* Multiple cooperative processes

Communicating with Preemptive Processes

No

- Interprocess variables
- Plug-ins
- **CALL PROCESS**
- **GET/SET PROCESS VARIABLE**
- **Begin/End SQL**

Yes

- Records
- Documents
- **HTTP Get**
- **CALL WORKER** (workers only)

Soft Spots

* Race conditions on files
* Illegal instructions sent via **CALL WORKER**
* Illegal instructions executed using any form of `eval()`