The New CICS Asynchronous API

Pradeep Gohil
Async API Technical lead
IBM UK

November 2018

Session GK

Please note…

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Agenda

1. Motivation for introducing the new CICS asynchronous API

2. Fundamentals of asynchronous processing for Application Developers

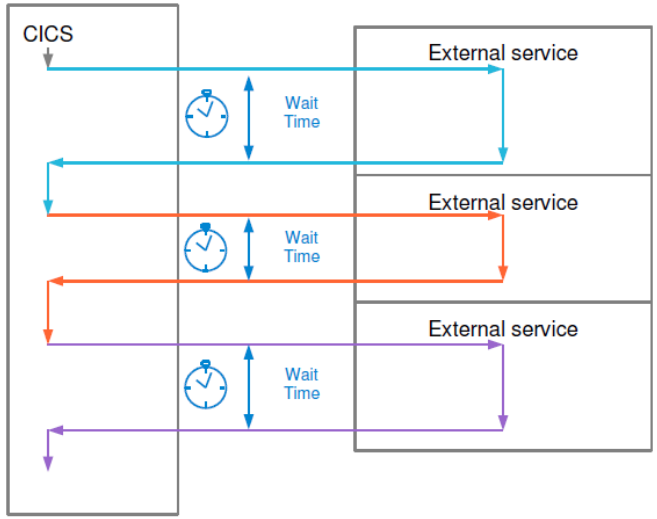
3. Control and management of asynchronous processing for System Programmers

Make better use of your time

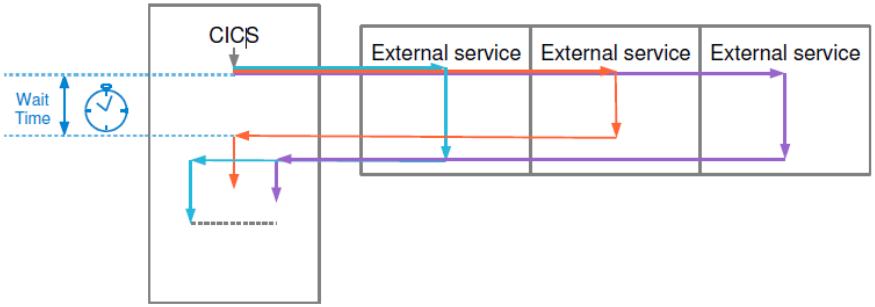
Could you do something more useful than waiting for a service call to return?

- Growing number of callable services
- Pressures to reduce application response times
- Greater resilience against unreliable services

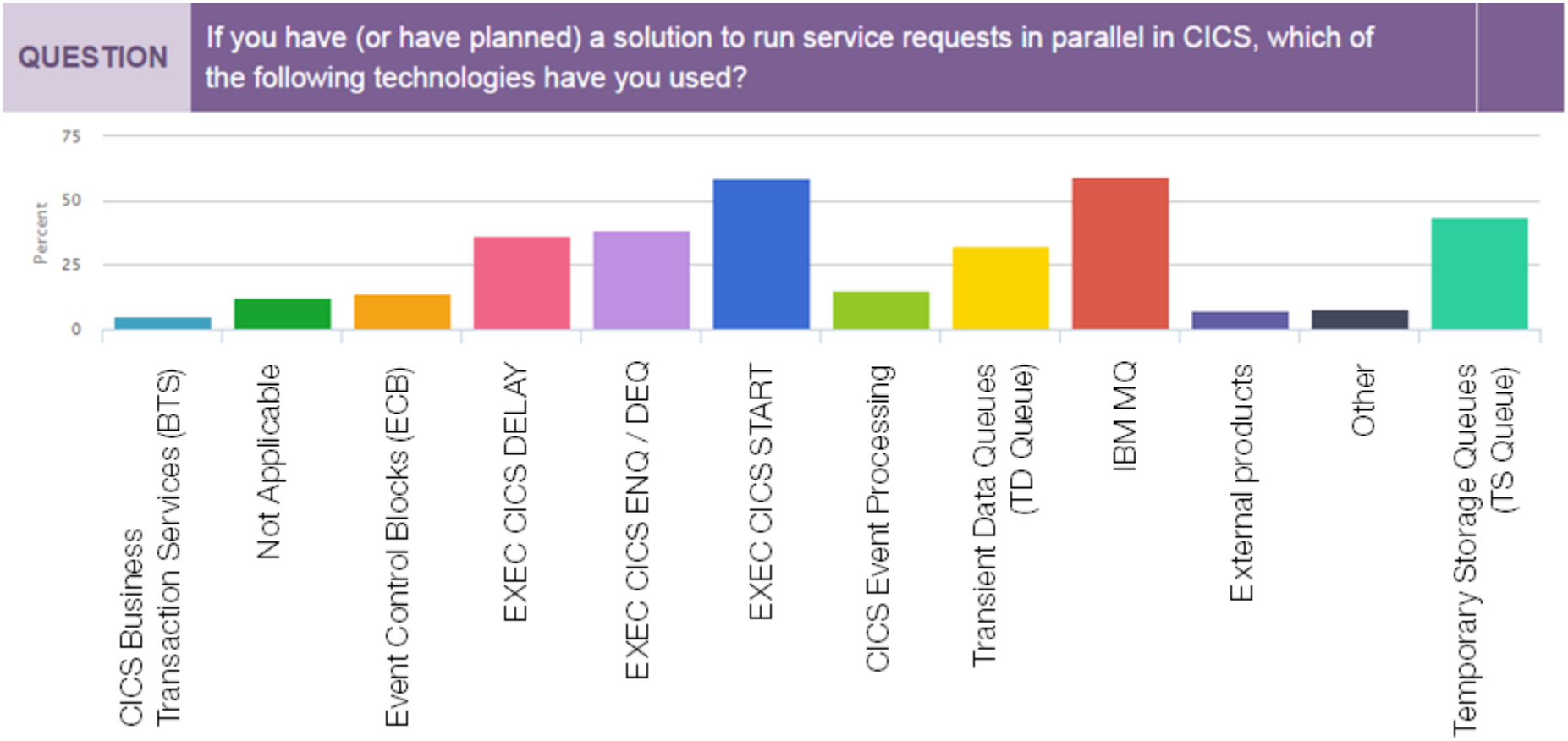
From this:



To this:



What are customers attempting today?



What are customers attempting today?

- No single clear solution
- Homegrown frameworks integrating different technologies together
- Difficulties with timing windows
- Require clean up transactions and ongoing management
- High cost of development and ongoing maintenance skills

What are customers attempting today?

"It would be difficult to implement my own solution..."

"Our current "solution in place" stinks but is just better than nothing.

IBM's solution needs to be better than ours but little imagination is needed to believe that will be the case. "

"Whilst a solution is in place, none are optimal.
Plenty of room to reduce missed sales opportunities"

"Asynchronous processing is quite alien to most application developers so you'll need a good way to handle this"

Features of the Asynchronous API

Timeouts

Parent-Child

Separate UoW

Task termination

JCICS

Policy

RUN TRANSID

Local children

Channel &
Containers

FETCH CHILD

FETCH ANY

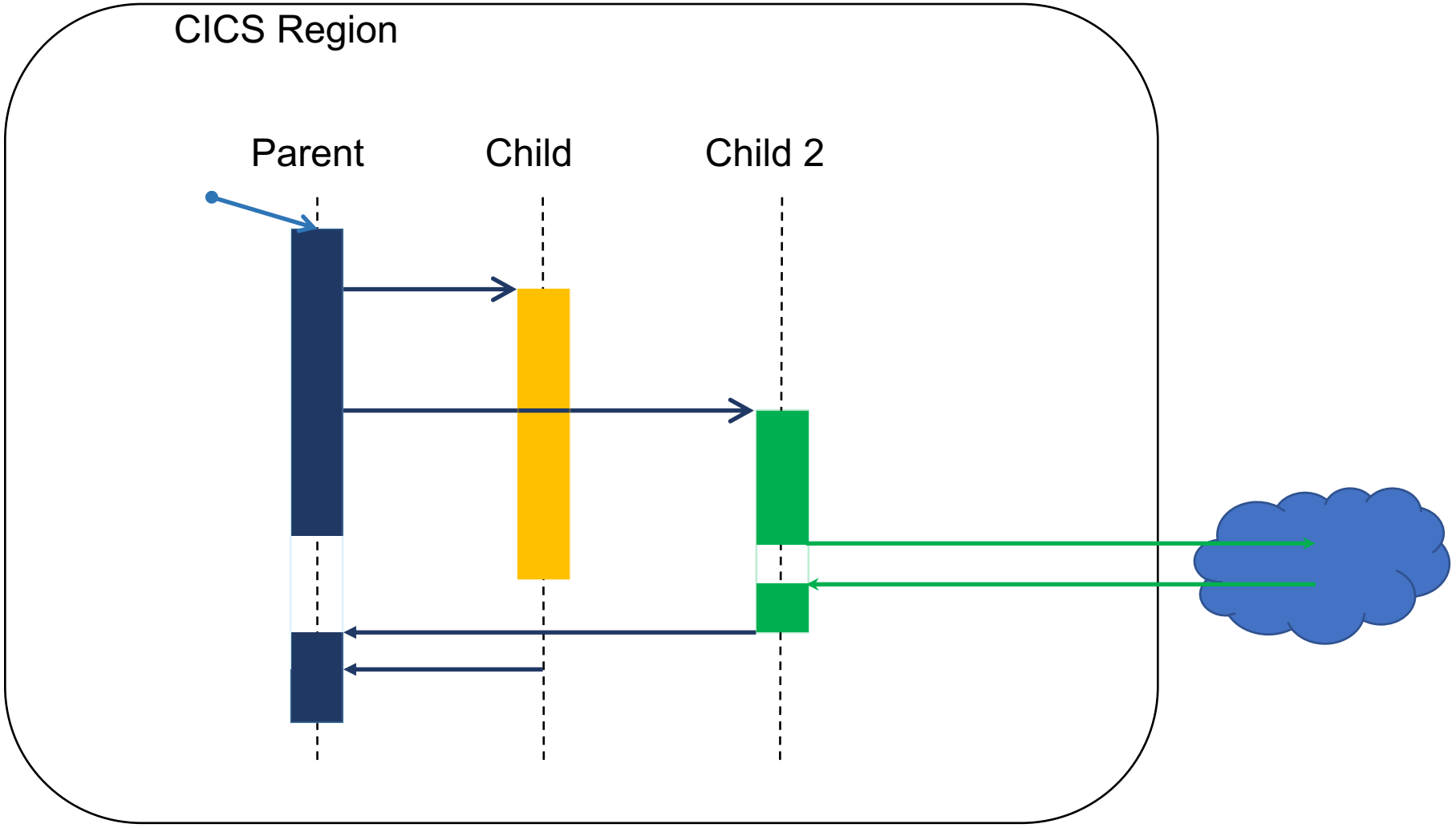
Management

FREE CHILD

Monitoring
& Statistics

Transaction Tracking

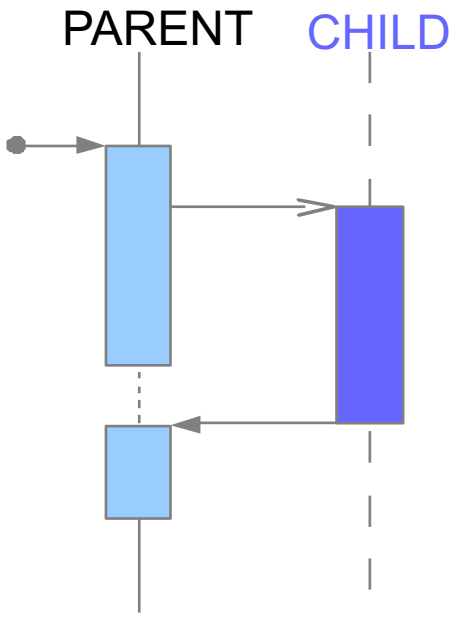
Parent – Child relationship



Aspects of asynchronous processing

Three Key Aspects

1. Run transaction asynchronously
2. Fetch child completion
3. Pass data safely



Features of the Asynchronous API

Timeouts

Parent-Child

Separate UoW

Task termination

JCICS

Policy

RUN TRANSID

Local children

Channel &
Containers

FETCH CHILD

FETCH ANY

Management

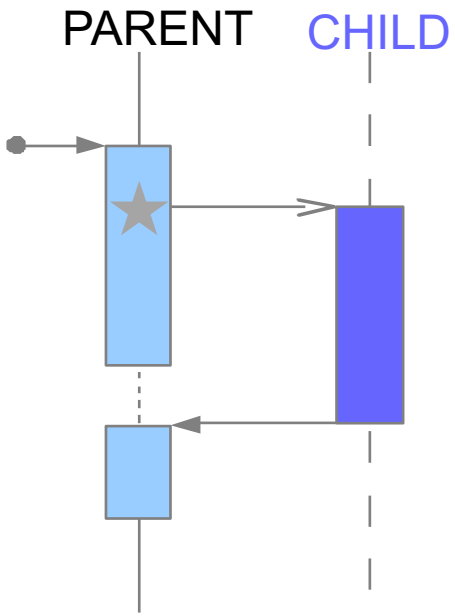
FREE CHILD

Monitoring
& Statistics

Transaction Tracking

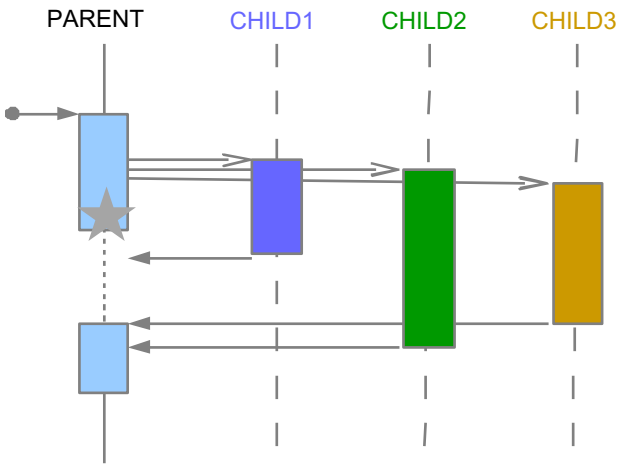
Run transactions asynchronously

```
RUN TRANSID (transaction)
  CHILD     (identifier)
```



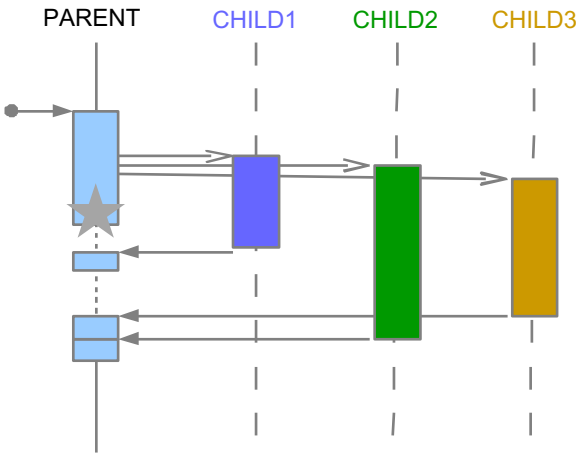
"I need a credit check but I can get on with some other work whilst I wait for the results"

Fetching child completion – Two new commands



`FETCH CHILD(in-identifier)`

"I cannot continue processing the parent coordinator, until I have received confirmation from CHILD3"

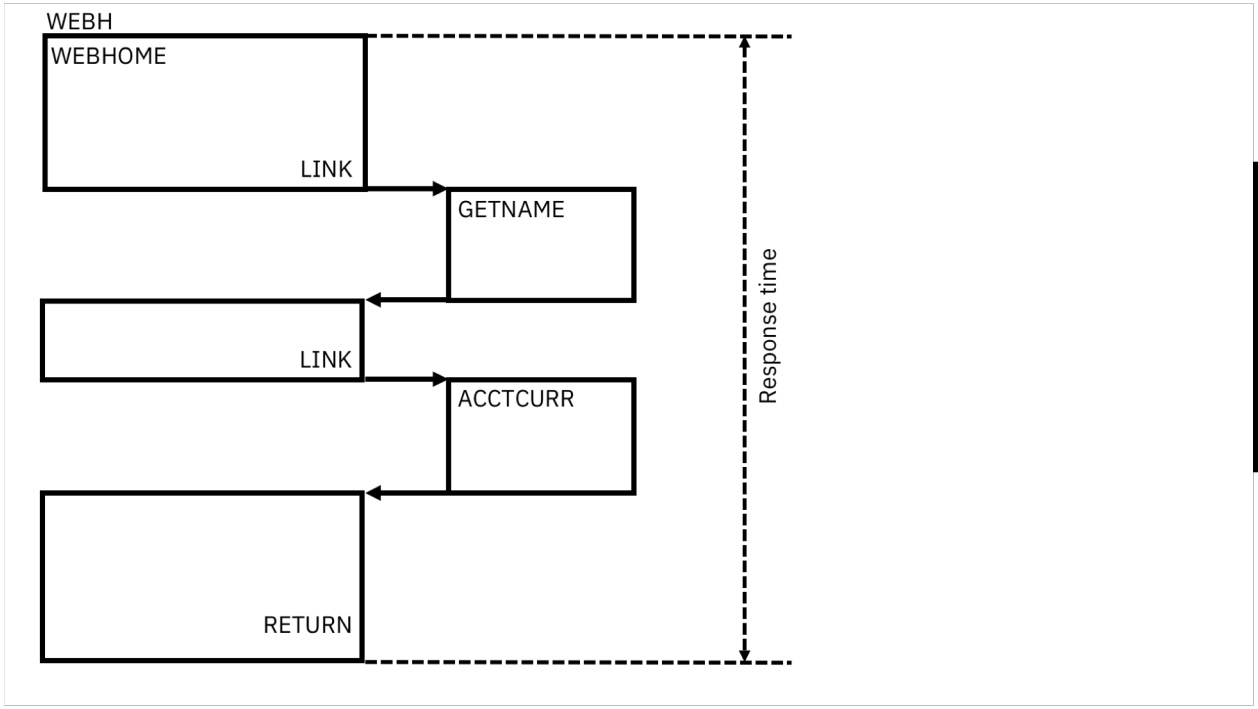


`FETCH ANY(out-identifier)`

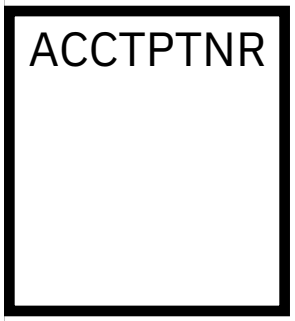
"I want to maximise the response time savings"
"I only need the first one to reply"

Use case: Extending applications whilst minimizing impact to response time

Existing business logic:

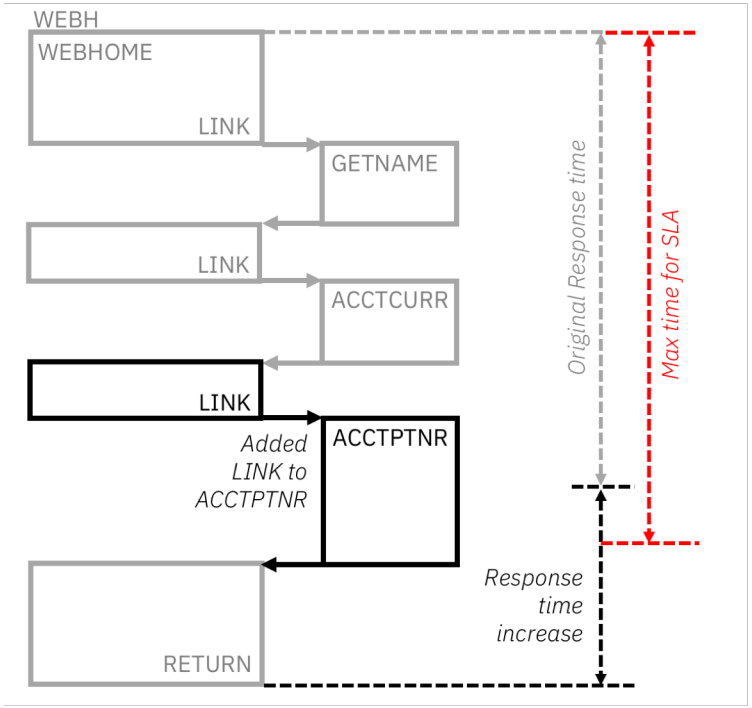


Extend with new service provider:

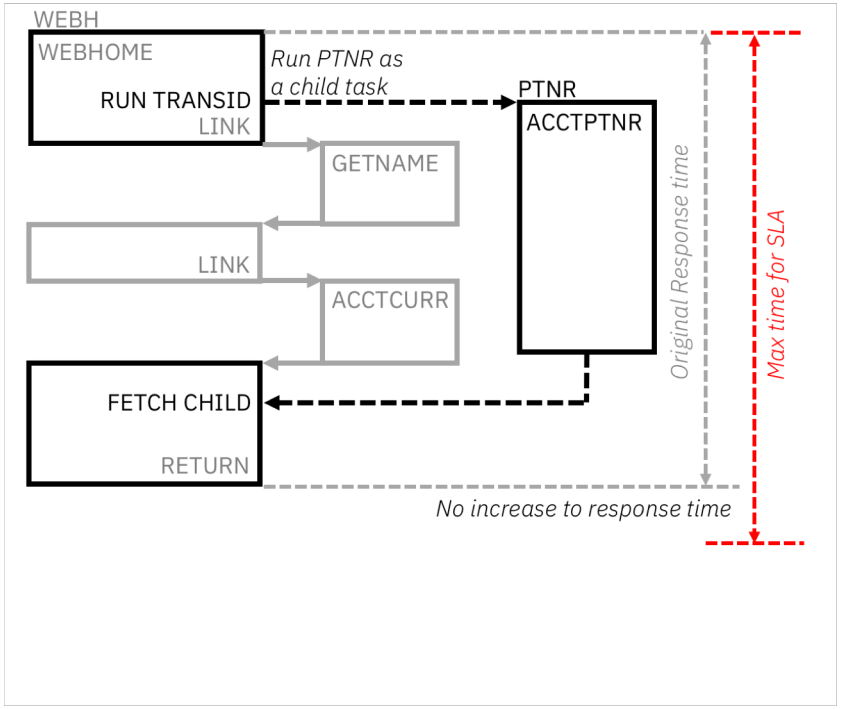


Use case: Extending applications whilst minimizing impact to response time

Typical solution:



Logic added asynchronously:

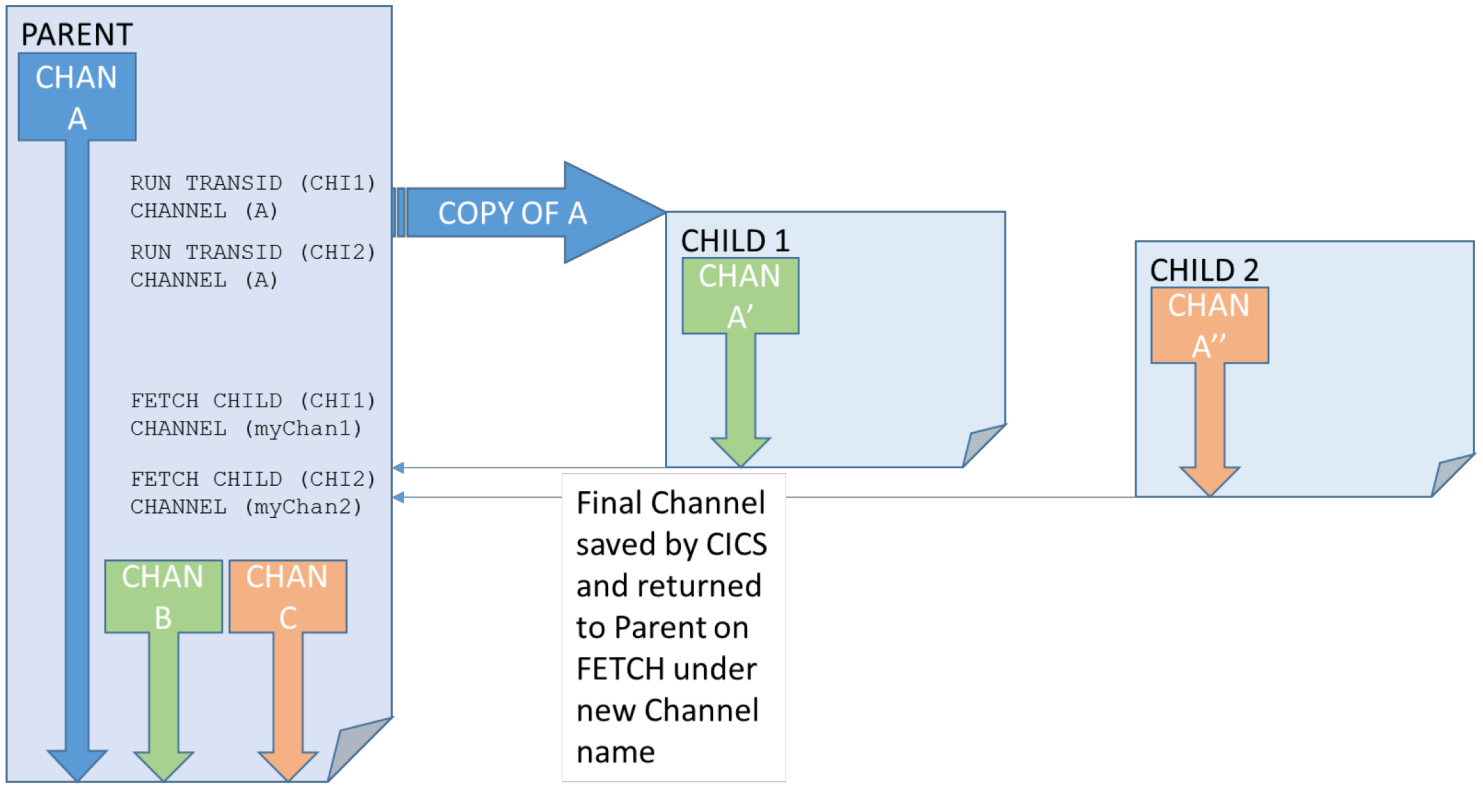


"Adding an outgoing call without affecting response time? Sounds impossible."

Passing data

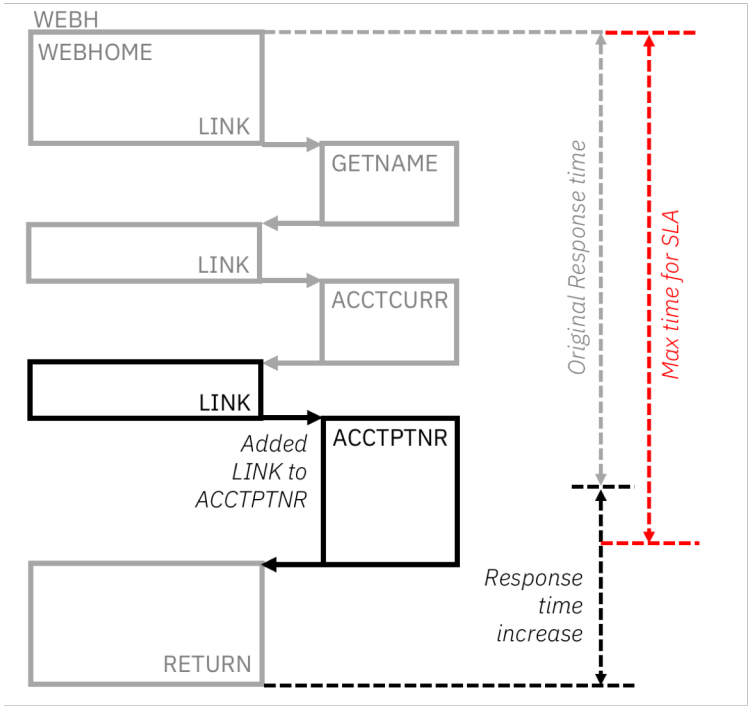
CICS Channels and Containers

(not COMMAREA)

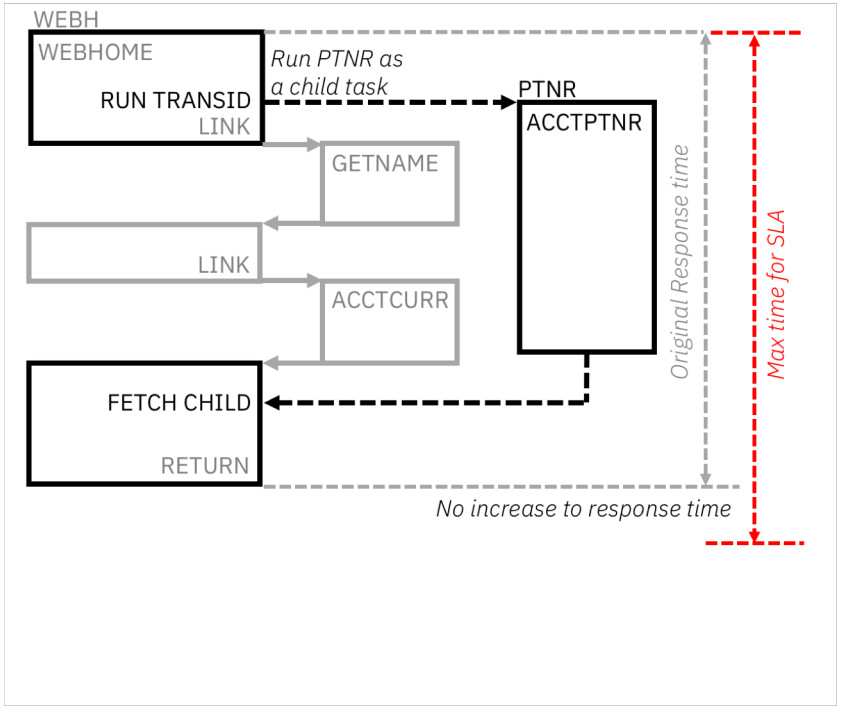


Use case: Extending applications whilst minimizing impact to response time – Revisited!

Typical solution:



Logic added asynchronously:



"Adding an outgoing call without affecting response time? Sounds impossible."

Use case: Extending applications whilst minimizing impact to response time – Revisited!

```
+          2 ACCTPTNR-CONTAINER    PIC X(16) VALUE 'ACCTPTNRCONT    '.

         1 MYCHANNEL               PIC X(16) VALUE 'MYCHANNEL       '.

           2 ACCTPTNR              PIC X(8) VALUE 'ACCTPTNR'.
           2 GETLOAN               PIC X(8) VALUE 'GETLOAN '.
+        1 TRANSIDS.
+          2 ACCTPTNR-TRAN         PIC X(4) VALUE 'PTNR'.
+
+        1 CHILD-TOKENS.
+          2 ACCTPTNR-TKN          PIC X(16).
+
+        1 RETURN-CHANNELS.
+          2 ACCTPTNR-CHAN         PIC X(16).
+
```

```
+        * ---------------------------------------------------------
+        * Asynchronously run PNTR to get account details
+        * from the partner bank
+        * ---------------------------------------------------------
+             EXEC CICS RUN TRANSID ( ACCTPTNR-TRAN )
+                           CHANNEL ( MYCHANNEL )
+                           CHILD   ( ACCTPTNR-TKN )
+                           RESP    ( COMMAND-RESP )
+                           RESP2   ( COMMAND-RESP2 )
+             END-EXEC
+
+             PERFORM CHECK-COMMAND
+
```

```
+        * ---------------------------------------------------------
+        * Get the customers current account details from the
+        * partner bank
+        * ---------------------------------------------------------
+             EXEC CICS FETCH CHILD      ( ACCTPTNR-TKN )
+                             CHANNEL    ( ACCTPTNR-CHAN )
+                             COMPSTATUS ( CHILD-RETURN-STATUS )
+                             ABCODE     ( CHILD-RETURN-ABCODE )
+                             RESP       ( COMMAND-RESP )
+                             RESP2      ( COMMAND-RESP2 )
+             END-EXEC
+
+             PERFORM CHECK-COMMAND
+             PERFORM CHECK-CHILD
+
+             EXEC CICS GET CONTAINER ( ACCTPTNR-CONTAINER )
+                           CHANNEL   ( ACCTPTNR-CHAN )
+                           INTO      ( PARTNER-ACCOUNTS )
+                           RESP      ( COMMAND-RESP )
+                           RESP2     ( COMMAND-RESP2 )
+             END-EXEC
+
+             PERFORM CHECK-COMMAND
+
+             PERFORM PRINT-PARTNER-ACCOUNTS-DETAILS
+
```

https://github.com/cicsdev/cics-async-api-redbooks/compare/start-of-3...end-of-3

Features of the Asynchronous API

Timeouts

Parent-Child

Separate UoW

Task termination

JCICS

Policy

RUN TRANSID

Local children

Channel &
Containers

FETCH CHILD

FETCH ANY

Management

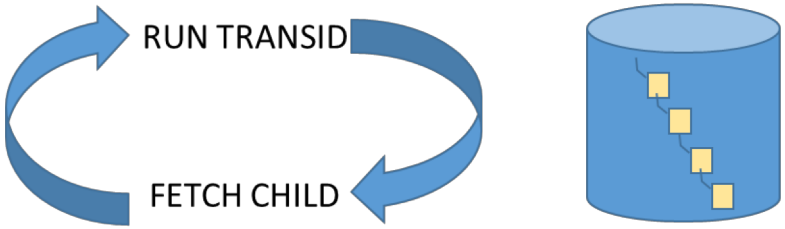
FREE CHILD

Monitoring
& Statistics

Transaction Tracking

FREE CHILD: Useful for long-running parents

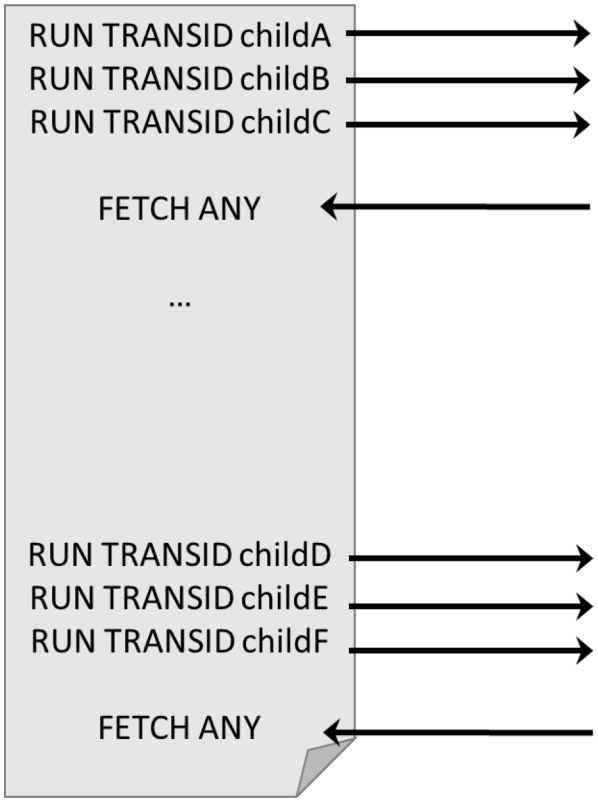
We keep child state information (just in case you ask for it later)



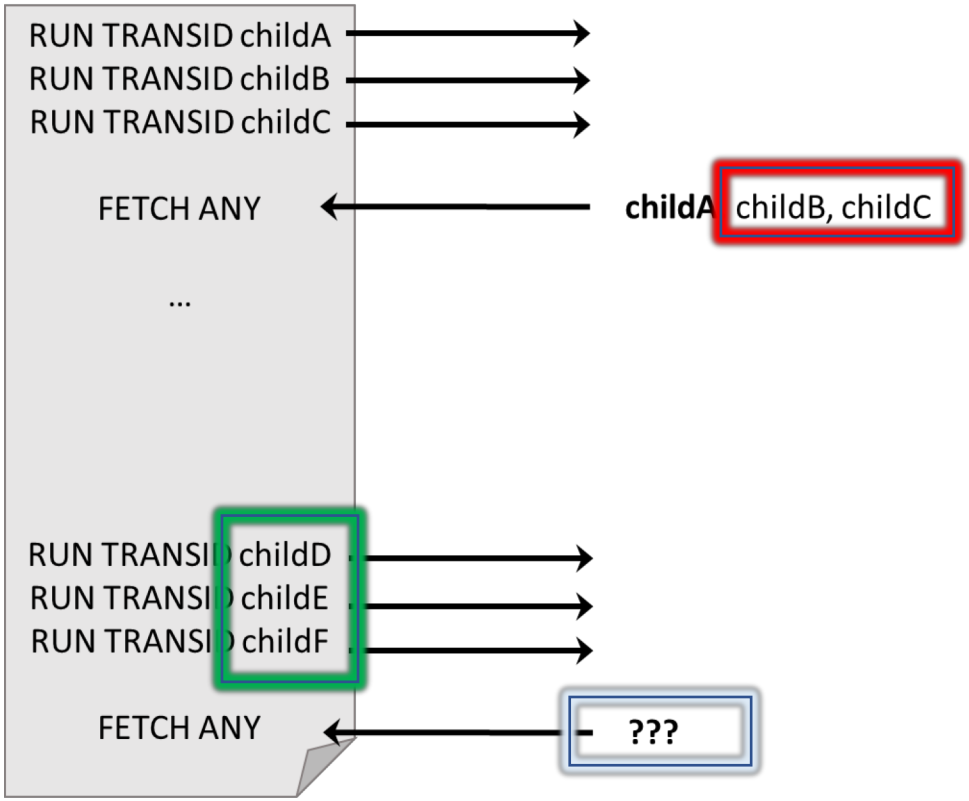
Ability to disassociate parent from child

App dev can DELETE CHANNEL for already FETCHed results

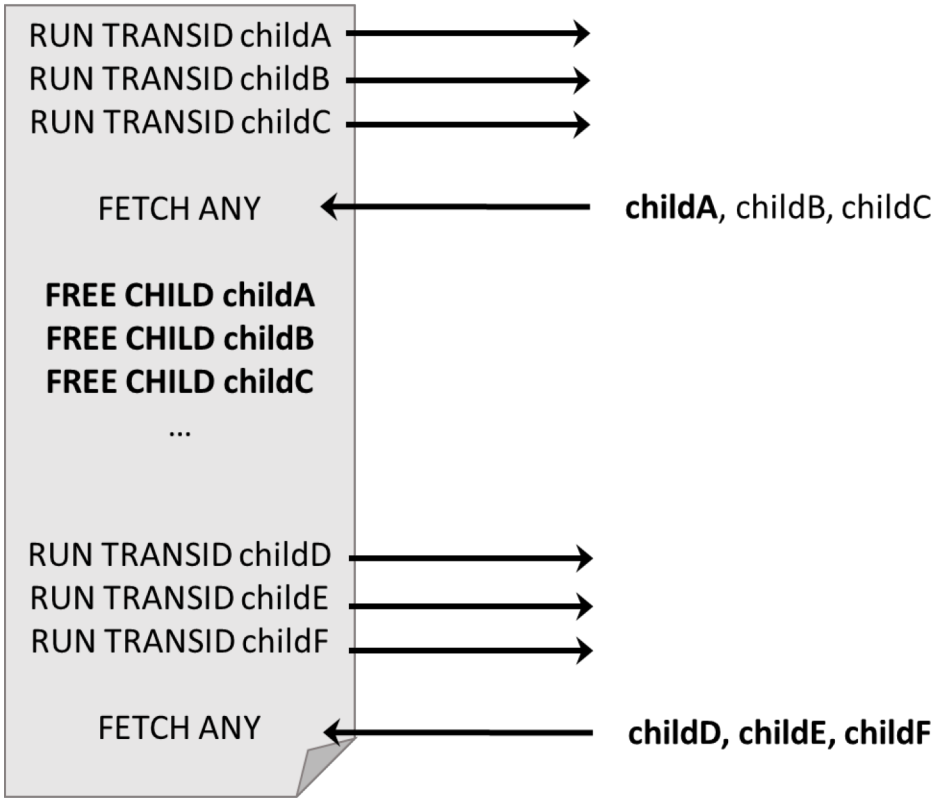
FREE CHILD: Sets of service calls



FREE CHILD: Sets of service calls



FREE CHILD: Sets of service calls



Features of the Asynchronous API

Timeouts

Parent-Child

Separate UoW

Task termination

JCICS

Policy

RUN TRANSID

Local children

Channel &
Containers

FETCH CHILD

FETCH ANY

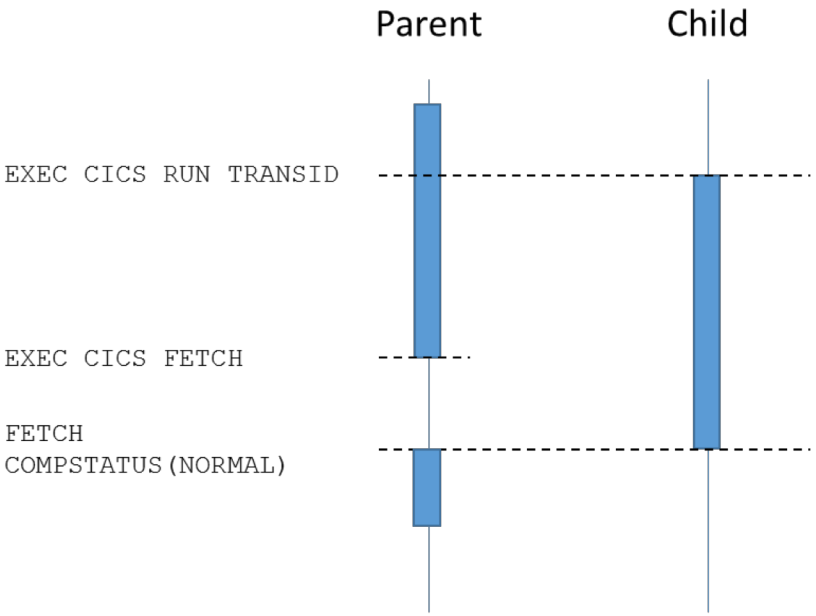
Management

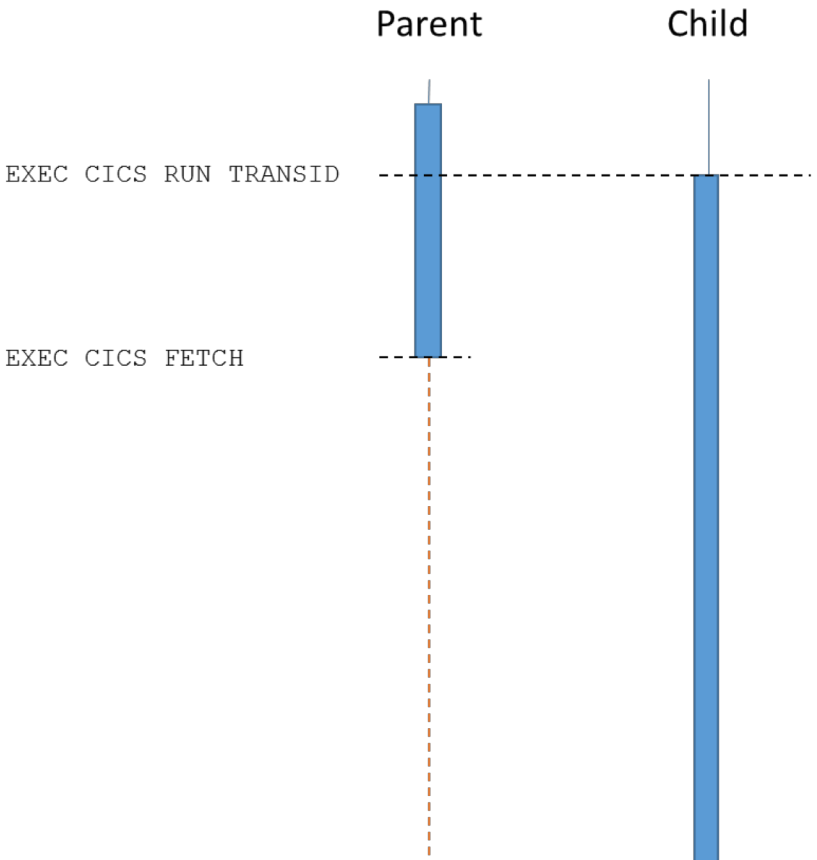
FREE CHILD

Monitoring
& Statistics

Transaction Tracking

Don’t want to wait forever!



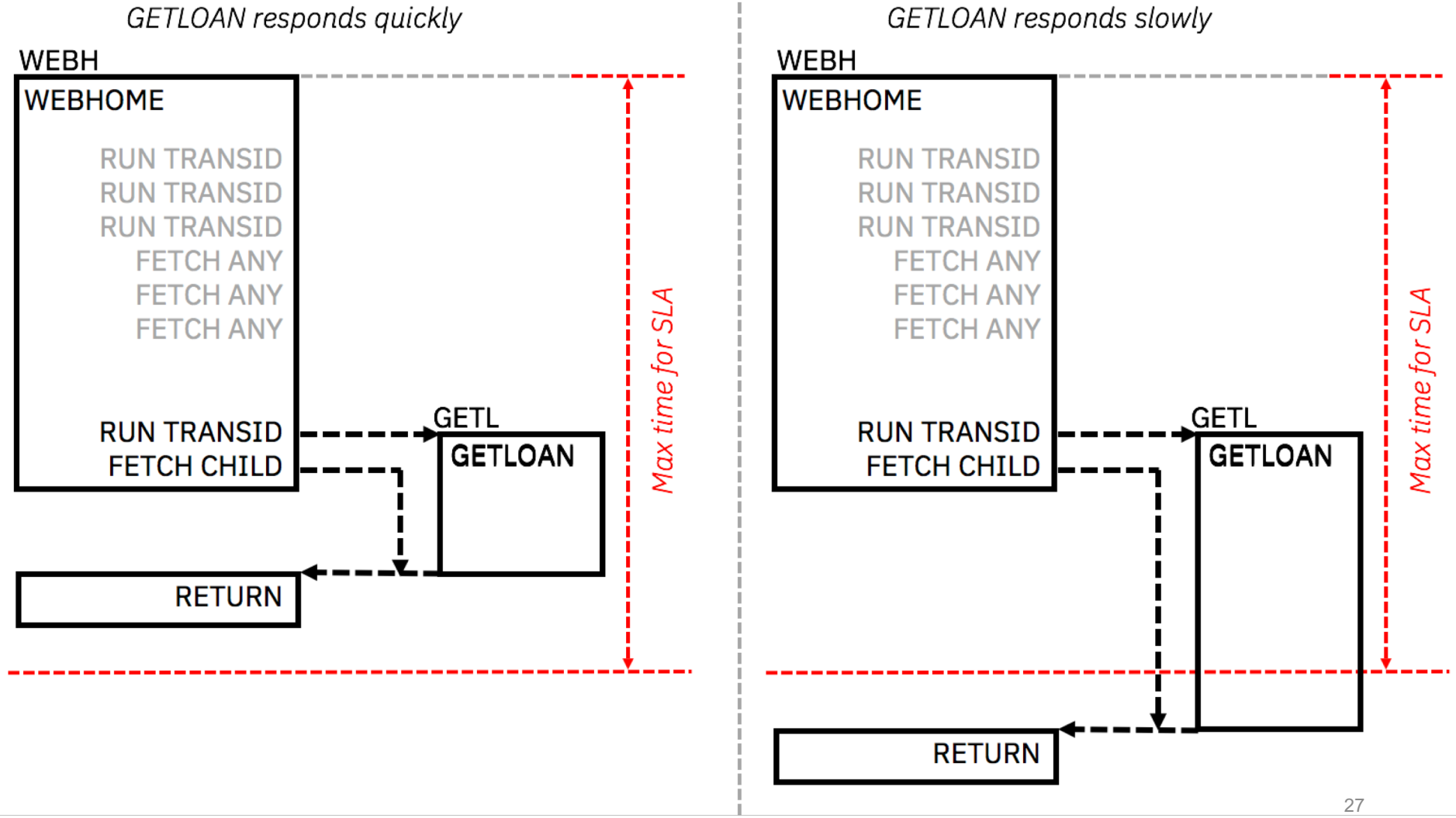


Don’t want to wait forever!

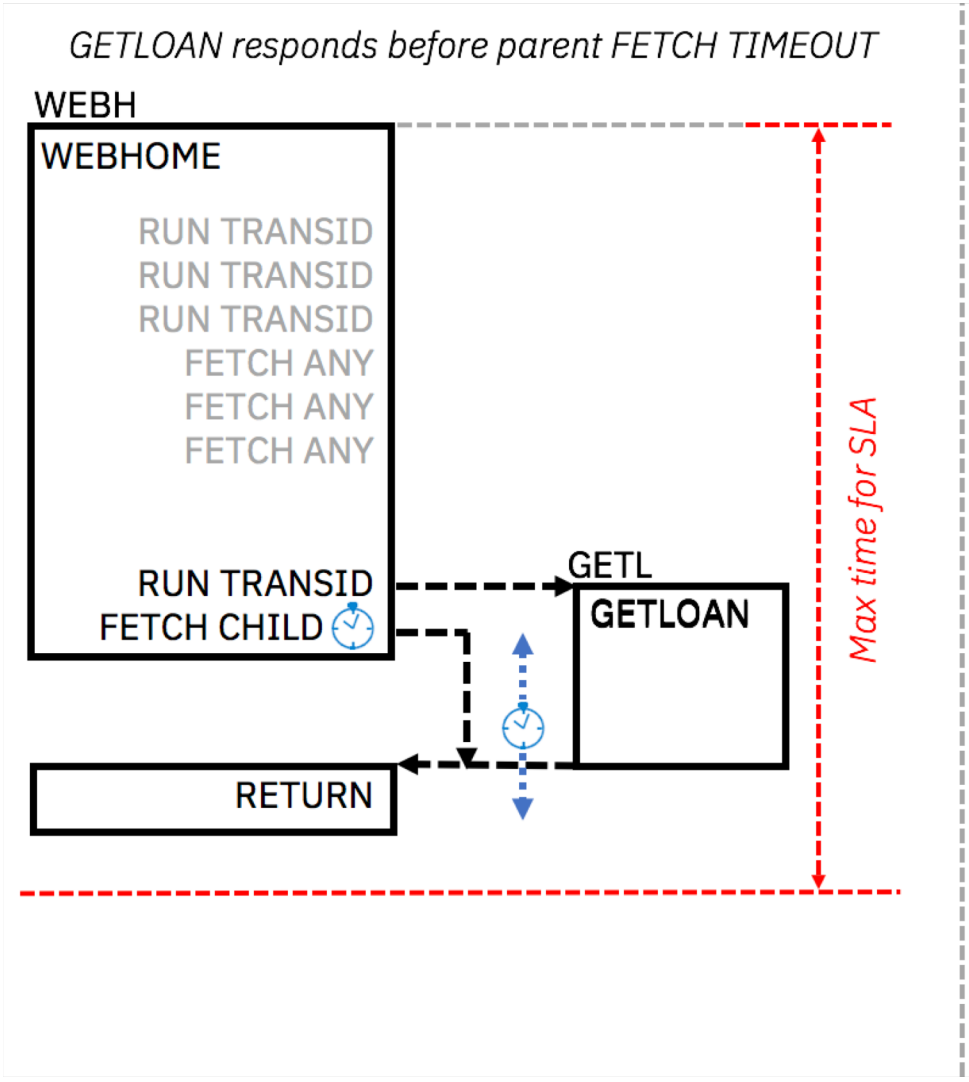
```
FETCH ANY
 >>-FETCH--ANY(data-area)--+--------------------+---------------->
                           '-CHANNEL(data-area)-'
 >--COMPSTATUS(cvda)--+-------------------+--------------------->
                      '-ABCODE(data-area)-'
 >--+---------------------+-----------------------------------><
    +-NOSUSPEND-----------+
    '-TIMEOUT(data-value)-'
```

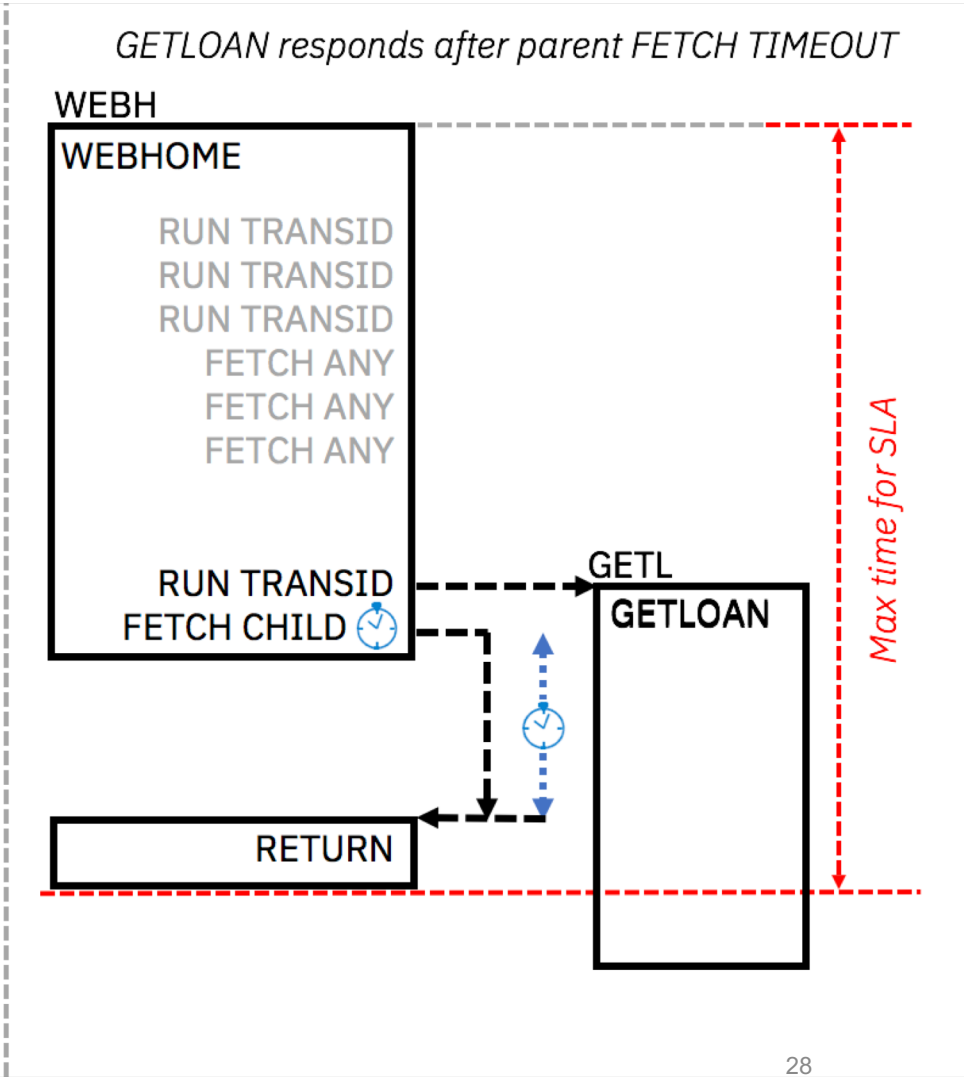
* TIMEOUT(0) means that timeout is not being set

Use case: Developing robust CICS applications with unreliable service providers



Use case: Developing robust CICS applications with unreliable service providers





RUN TRANSID

```
RUN TRANSID

>>-RUN--TRANSID(name)--+---------------+--CHILD(data-area)-----><
                       '-CHANNEL(name)-'
```

RESP CODES:

CHANNELERR
INVREQ
TRANSIDERR
NOTAUTH
DISABLED

FETCH CHILD

```
FETCH CHILD

>>-FETCH--CHILD(data-value)--+--------------------+------------->
                            '-CHANNEL(data-area)-'

>--COMPSTATUS(cvda)--+-------------------+---------------------->
                     '-ABCODE(data-area)-'

>--+---------------------+-------------------------------------><
   +-NOSUSPEND-----------+
   '-TIMEOUT(data-value)-'
```

COMPSTATUS:

ABEND
NORMAL
SECERROR

RESP CODES:

NOTFINISHED
INVREQ

FETCH ANY

```
FETCH ANY

>>-FETCH--ANY(data-area)--+--------------------+---------------->
                          '-CHANNEL(data-area)-'

>--COMPSTATUS(cvda)--+-------------------+---------------------->
                     '-ABCODE(data-area)-'

>--+---------------------+-------------------------------------><
   +-NOSUSPEND-----------+
   '-TIMEOUT(data-value)-'
```

COMPSTATUS:

ABEND
NORMAL
SECERROR

RESP CODES:

NOTFINISHED
INVREQ
NOTFND

Features of the Asynchronous API

Timeouts

Parent-Child

Separate UoW

Task termination

JCICS

Policy

RUN TRANSID

Local children

Channel &
Containers

FETCH CHILD

FETCH ANY

Management

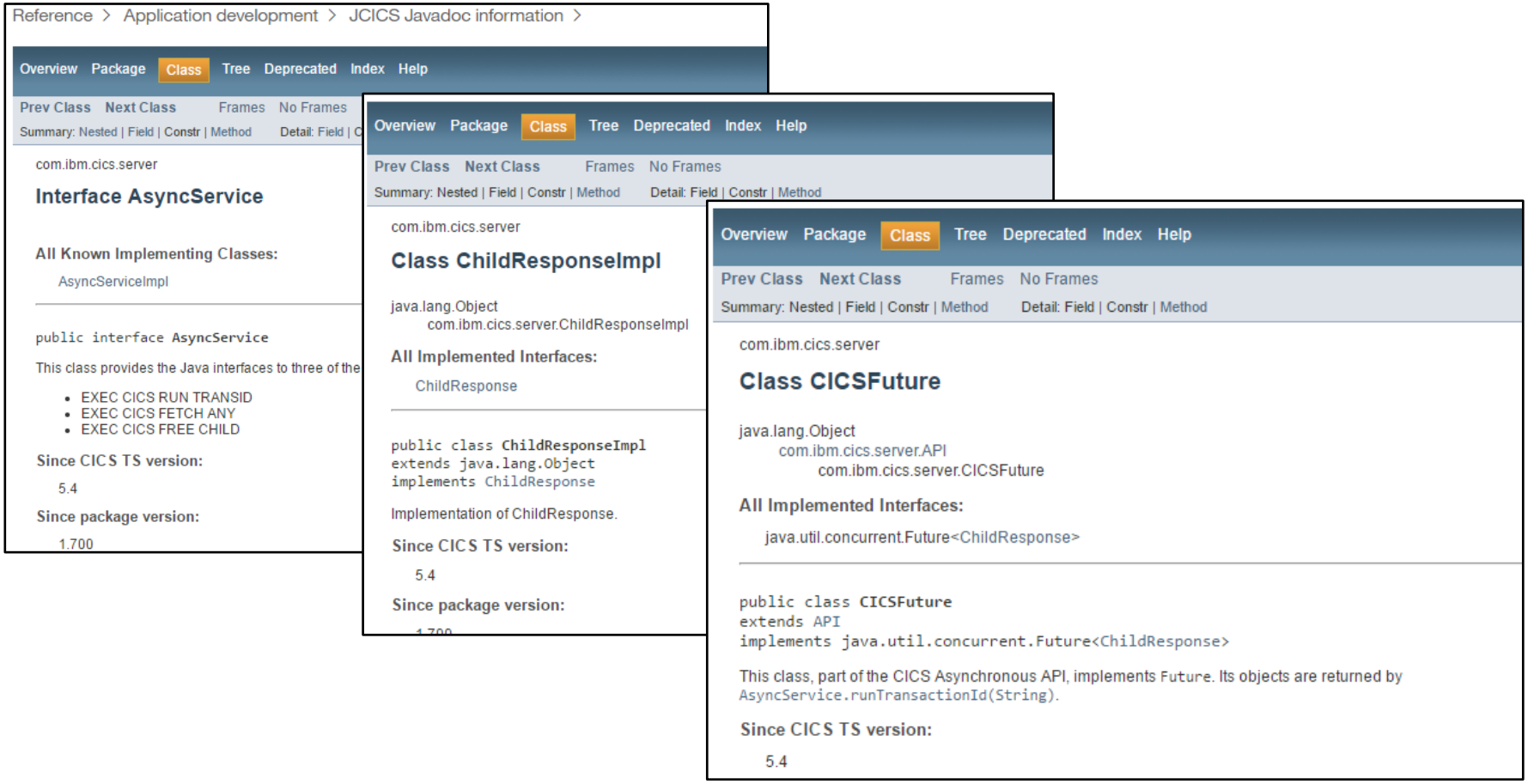
FREE CHILD

Monitoring
& Statistics

Transaction Tracking

JCICS variant of Asynchronous API commands

To complement a Java style of coding



```
import java.util.concurrent.Future;
import com.ibm.cics.server.AsyncService;
import com.ibm.cics.server.AsyncServiceImpl;

AsyncService async = new AsyncServiceImpl();

Future<ChildResponse> loanRate = null;
loanRate = async.runTransactionId(getLoanTran, myChannel);


…



ChildResponse returnedLoanRate = null;
returnedLoanRate = loanRate.get(500, TimeUnit.MILLISECONDS);

if
(returnedLoanRate.getCompletionStatus().equals(CompletionStatus.NORMAL))
        {
                …
        }
```

JCICS Implementation

- Standard Java: Future interface
- Familiar to Java programmers
- Complete EXEC CICS functionality
- Interfaces and implementations to allow for mocking

Features of the Asynchronous API

Timeouts

Parent-Child

Separate UoW

Task termination

JCICS

Policy

RUN TRANSID

Local children

Channel &
Containers

FETCH CHILD

FETCH ANY

Management

FREE CHILD

Monitoring
& Statistics

Transaction Tracking

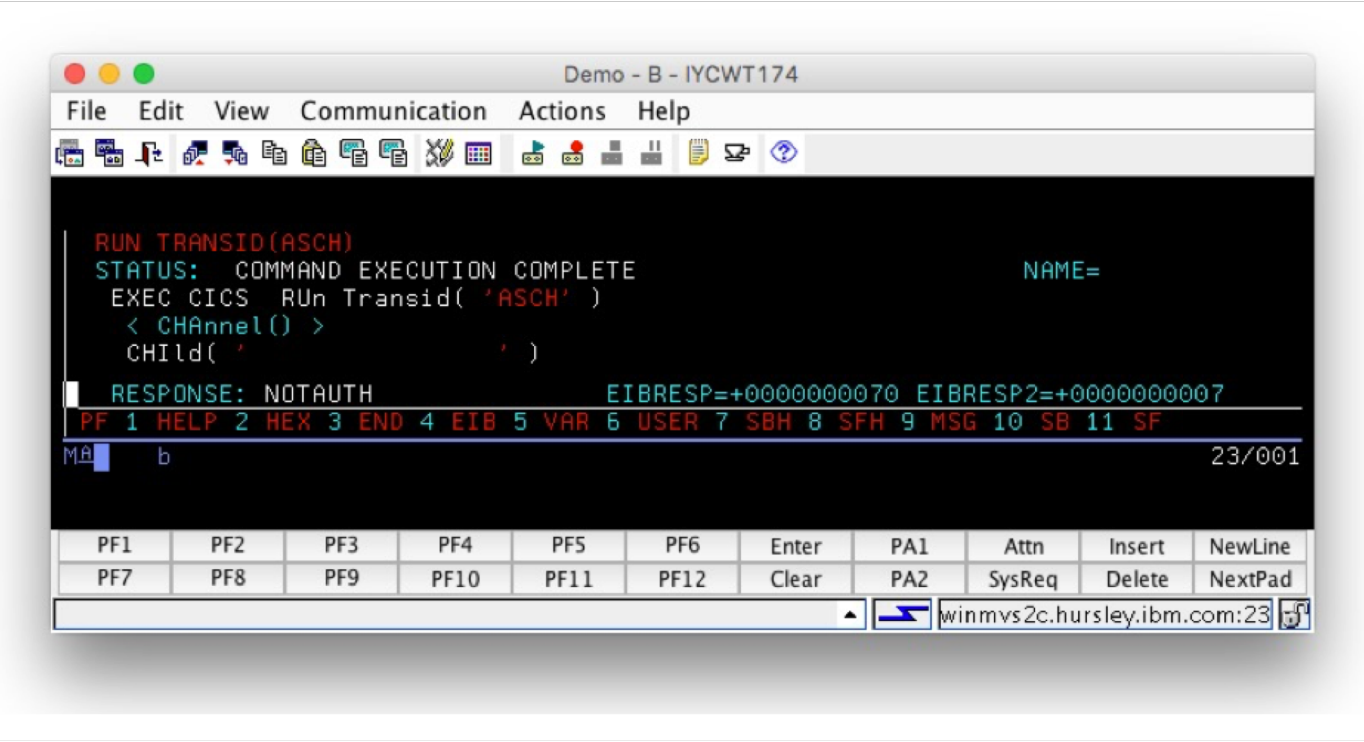
CICS automated control

- Emergency Brake

- Control dispatching of asynchronous tasks
- Protect CICS rather than to optimise throughput
- Prevent flooding the CICS system
- Managed by the CICS system

Security

Security context for child inherited from parent



Transaction Classes

Control your systems!

✓ **DO** have transaction classes

✘ **DO NOT** put parents and children in the same transaction class

Dumps

- New CICS ‘AS’ domain provides enhancements to
  - Systems dumps

```
===> VERBX DFHPD710 'AS=3'
```

  - Transaction dumps

```
ASYNCHRONOUS CHILD SUMMARY FOR THE CURRENT TRANSACTION
 TOTAL NUMBER OF CHILDREN CREATED : 00000005
 TOTAL NUMBER OF CHILDREN FREED   : 00000001
ASYNCHRONOUS CHILD LIST
Child_Trannum(00046) Child_Tranid(TRB2) Child_State(ABENDED   ) Child_Token(00000050_41800048 , 0000046C_00000001)
Child_Trannum(00047) Child_Tranid(TRB3) Child_State(ACTIVE    ) Child_Token(00000050_418000C8 , 0000047C_00000002)
Child_Trannum(00048) Child_Tranid(TRB4) Child_State(COMPLETED ) Child_Token(00000050_41800148 , 0000048C_00000003)
Child_Trannum(00049) Child_Tranid(TRB5) Child_State(NOTSTARTED) Child_Token(00000050_418001C8 , 0000049C_00000004)
```

```
This transaction is running as an ASYNCHRONOUS child task
Child_Token(00000050_41800048 , 0000046C_00000001)
Parent_Trannum(00045)     Parent_Tranid(TRB1)           Parent_State(ACTIVE   )
```

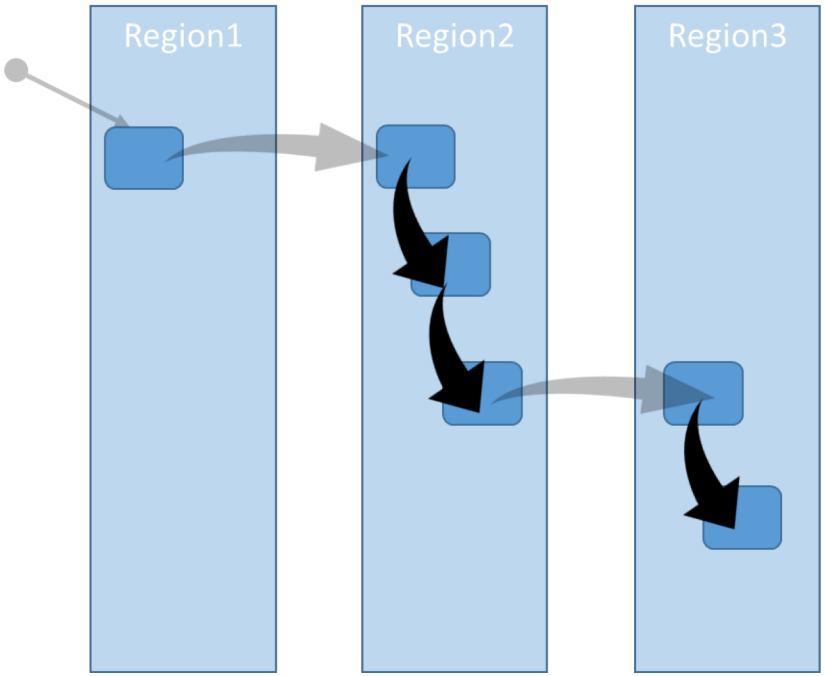
Monitoring and Statistics

◢ Clocks and timings		
AS Fetch Wait Time	ASFTCHWT	0000:00:10.651358
AS Run Delayed Time	ASRNATWT	0000:00:00.000000
◢ Request counts		
EXEC CICS FETCH Count	ASFTCHCT	21
EXEC CICS FREE CHILD Count	ASFREECT	7
EXEC CICS RUN TRANSID Count	ASRUNCT	18
Total Asychronous API Commands	ASTOTCT	46

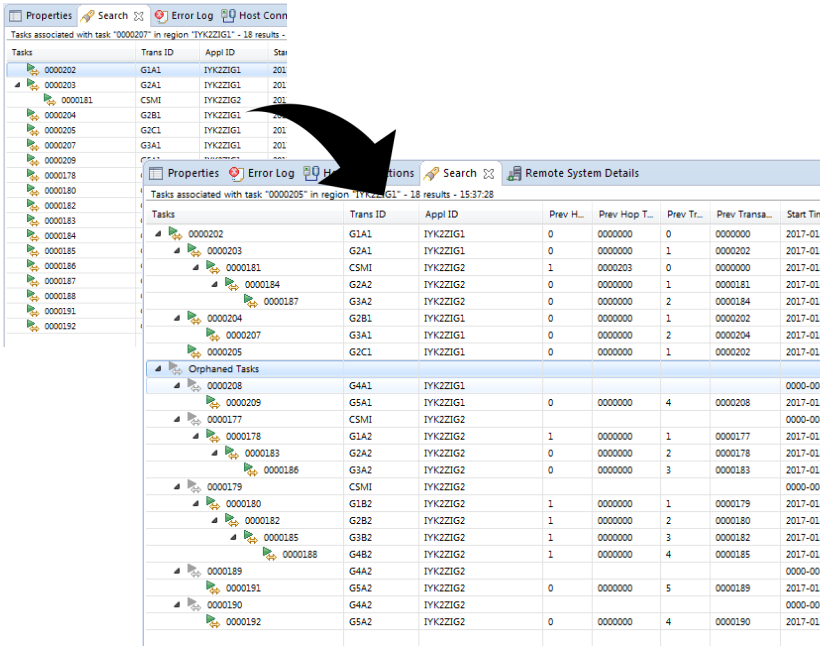
```
          Sample Program - CICS Statistics Print Report Selection
                                                  06/02/2017  12:37:59

Select the statistics reports required and press 'Enter' to validate

 DB2 Connection . . . . . N   WebSphere MQ Connection. . . . . N
 DB2 Entries. . . . . . . N   WebSphere MQ Monitors. . . . . N
                              Program
 Asynchronous Services. . . Y Terminal
                              Connecti
                              TCP/IP .
```

```
Asynchronous services
_____________________

  RUN commands. . . . . : 4,026
  FETCH commands. . . . . :    11
  FREE commands . . . . . :     0
  Current active children . . . . . :     0
  Peak active children. . . . . :     7
  Times RUN commands being delayed. . :   744
  Current parents being delayed . . . :     0
  Peak parents being delayed. . . . . :     1
```

Transaction Tracking




```
PTCOUNT(data-area)
PTSTARTTIME(data-area)
PTTASKID(data-area)
PTTRANSID(data-area)
```



Features of the Asynchronous API

Timeouts

Parent-Child

Separate UoW

Task termination

JCICS

Policy

RUN TRANSID

Local children

Channel &
Containers

FETCH CHILD

FETCH ANY

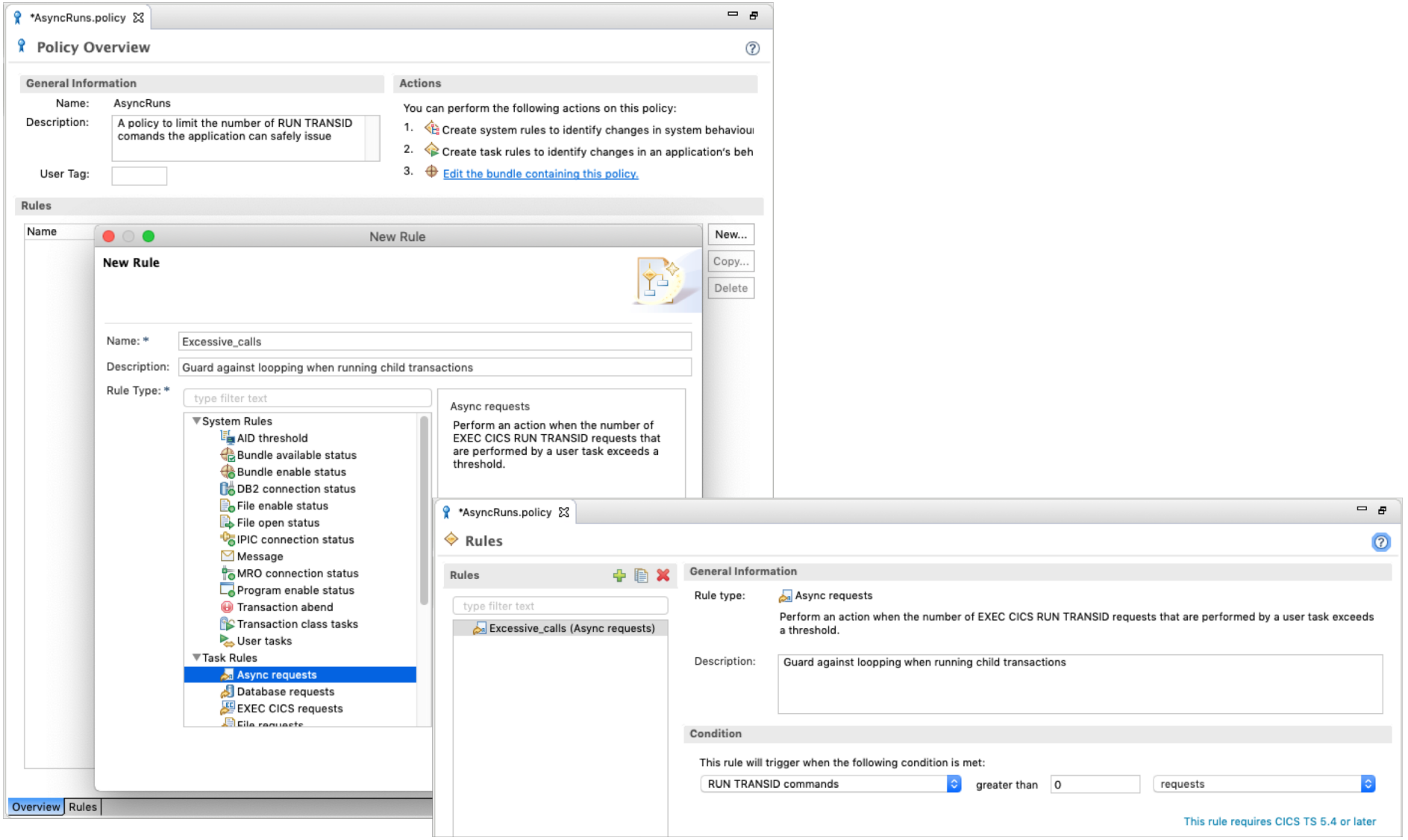
Management

FREE CHILD

Monitoring
& Statistics

Transaction Tracking

Async Requests Policy



Articles, IBM Knowledge Center, Example code

Search **CICS Asynchronous API**

github.com/**cicsdev**

cics-async-api-fetch-child-example

A basic example demonstrating the passing of information from a parent to a child program using the CICS asynchronous API.

C Updated on 5 Oct 2016

cics-async-api-credit-card-application-example

An example application that compares calling services sequentially versus asynchronously, using the new CICS asynchronous API.

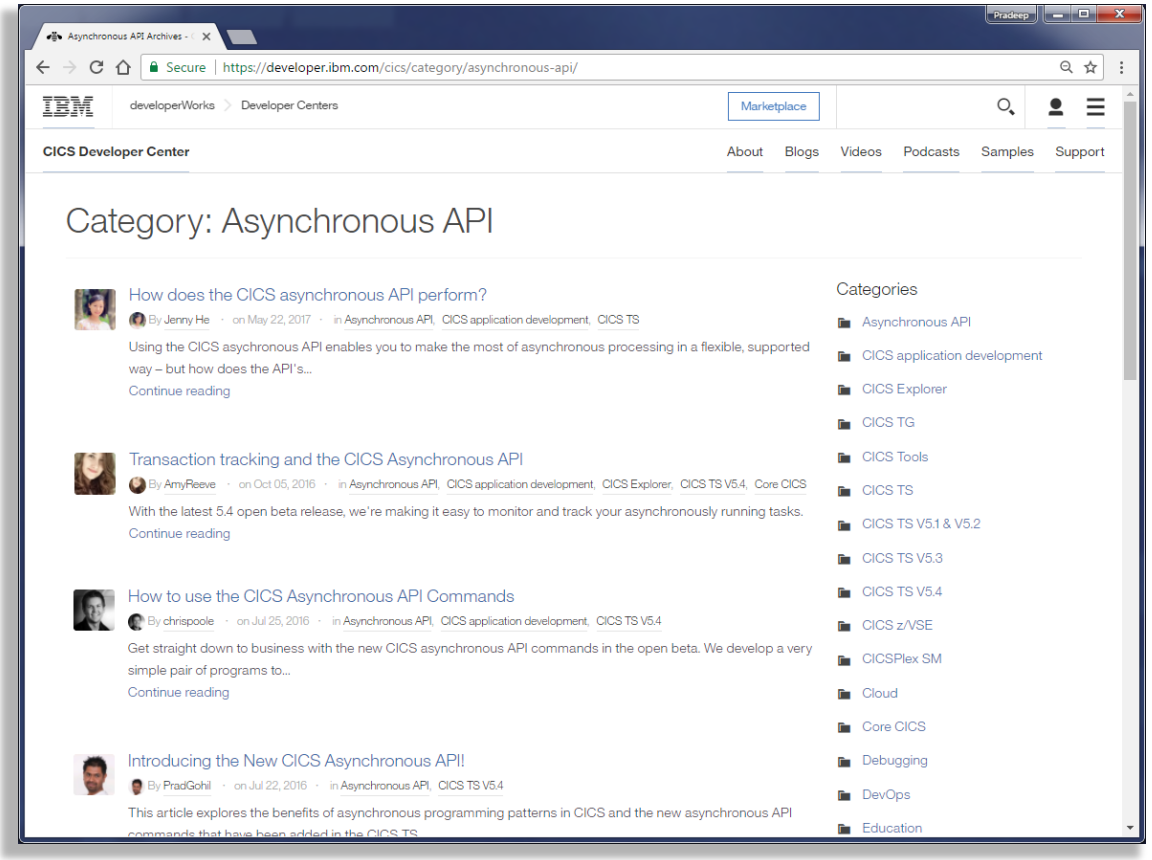
COBOL Updated on 5 Oct 2016

cics-async-api-channel-usage-example

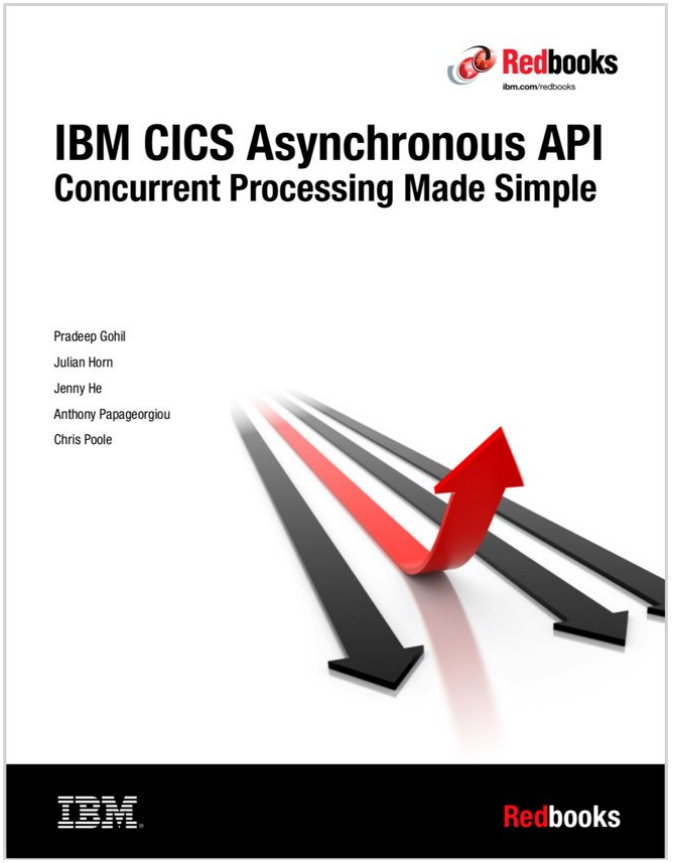
Async API channel usage example

Assembly 1 Updated on 20 Oct 2016

developer.ibm.com/cics/category/asynchronous-api/



Available NOW!



Fully detailed examples on how to achieve:

- Enhanced functionality
- Reduced Response Times
- Greater Robustness

Examples in COBOL and Java on GitHub
https://github.com/cicsdev/cics-async-api-redbooks

Tips and Tricks on interesting ways to use the API

Detail about performance and monitoring

Download here:
http://www.redbooks.ibm.com/abstracts/sg248411.html?Open

Processing in Seconds

made with Z story

Walmart enhance their complex search capabilities by harnessing asynchronous processing to achieve large-scale I/O in minimal time. The result is a transformed response time from 2 minutes, to only 1 second.

Made with Z Stories: Walmart, North America

THE BIG IDEA

Walmart's customer-facing application provides search results for key events from big data. Their challenge is to reduce an average 2 minute response time, to meet an SLA of … just 1 second!

Walmart's global distributed platform generates and records 500M events per day, driving over 1000 tps on Z. This large data is interrogated by a sophisticated search mechanism with response time averages of 2 minutes.

Harnessing the I/O processing capabilities on Z, Walmart undertook a project to transform a serially-executing search algorithm and utilized asynchronous programming techniques to dramatically reduce the response time.

TAKING ACTION

Walmart's serially executing search algorithm was benchmarked at 5,000 file reads/sec.
Using native VSAM on Z saw an increase to 60,000 reads/sec. Though this still wasn't good enough.

Appreciating the bottleneck on File I/O, Walmart leveraged the CICS Asynchronous API to spread the file operations across numerous tasks. Run concurrently, these tasks were able to achieve truly high volume transaction rates and increase I/O capabilities.

Employing the CICS Asynchronous API, Walmart greatly simplified the complexity of writing their own homegrown asynchronous patterns. Reducing the maintenance overhead and lines-of-code, they also improved the application's fault tolerance and resilience.

RESULT!

Walmart were very impressed with their enhanced complex-search capabilities. They transformed an application capable of 5,000 reads/sec to 700,000 reads/sec, by employing asynchronous processing techniques.

They remarked "It just would not have met the SLA doing it serially". Coupled with the CICS Asynchronous API, they added:

"This is really a cool service… all we're doing is using what CICS has given us; we're just doing it in a creative way."

In a sentence? Using asynchronous processing to achieve large scale I/O in minimal time

Last update: July 2018

We want your feedback!

- Please submit your feedback online at ….
  - http://conferences.gse.org.uk/2018/feedback/gk
- Paper feedback forms are also available from the Chair person
- This session is GK