#SHAREorg

SHARE
Technology · Connections · Results

Improving the performance of web-initiated CICS transaction workloads

Patrick Fournier
Serge Bourlot
John Bachiochi
(SysperTec)

Tuesday, August 13, 2013 (12:15 to 1:15 PM)
Session # 13456

SHARE in Boston

Presenters' Bio

- Patrick Fournier
  - Modernization of IBM mainframe applications
  - Last 5 years: web-enablement of 3270 applications
  - Initiator of CICS performance improvement R&D project
- Serge Bourlot
  - z/OS system product developer (custom TS)
  - Strong z/OS internals - especially multi-tasking services
  - Lead developer for R&D project
- John Bachiochi
  - Develop and support VSE and MVS SW products
  - Part of benchmarking team for R&D project

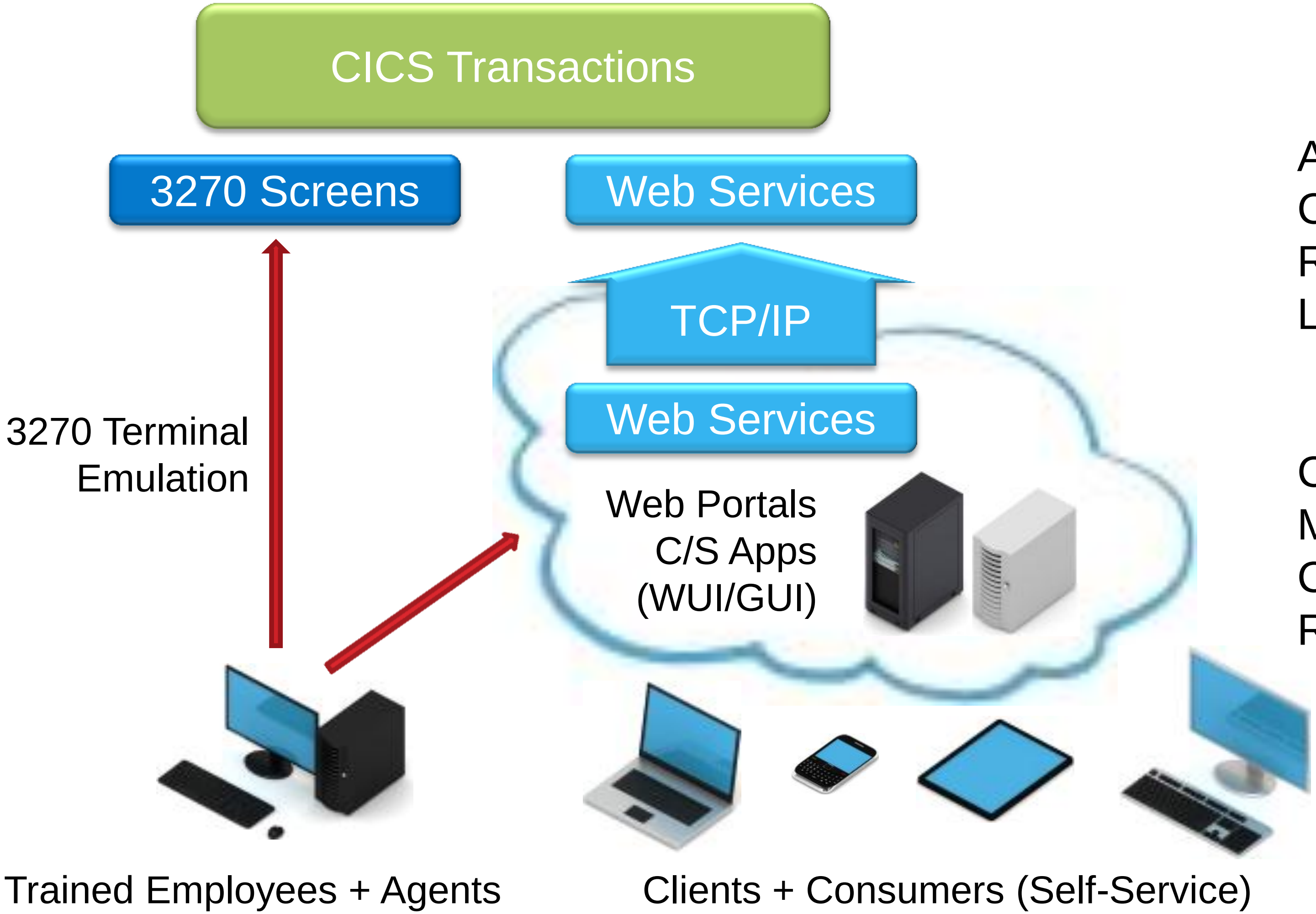
Changing Nature of CICS TP Workloads

A growing number of consumers conduct their personal business and shop in the Cloud every day, which results in enormous flows of web-initiated transactions hitting the supporting – oftentimes CICS – applications.

How It All Started …

- Two web-enablement prospects with similar exponentially-growing web-initiated CICS transaction workloads:
  - Improve web access (middleware) to CICS apps
  - Tens of million of CICS transactions per day
  - Web-initiated traffic reaching CICS apps as web services
  - Clients + consumers in “self-service” mode via web portals
  - Small number of query transactions →85% TP workload
  - No direct revenue generation: part of expected “service”
  - Cost of doing business today: improved user experience

Changing Nature of CICS TP Workloads



+

Accessibility
Convenience
Revenues
Labor costs

\−

CPU Usage
Mainframe costs
Capacity
Response times

Transaction Processing: Past, Present, and Future (IBM Redbooks redp4854)

“We run several thousand transactions per second now that our systems are opened up to the Internet and mobile devices.”

“The number of look-to-book transactions for our hotel chain has changed from 8 – 10 searches for one booking to potentially thousands of searches for one booking.”

“We expect more growth coming from the mobile channel and we also foresee a workload increase from new self-service applications.”

“In our enterprise architecture, the mainframe is our transaction processing box; it is optimized for transaction processing, and we expect it to be further optimized for this in the future.”

We have moved most of our services to stateless web services, which expose core transactions directly.”

CICS: Original Design vs. Current Usage

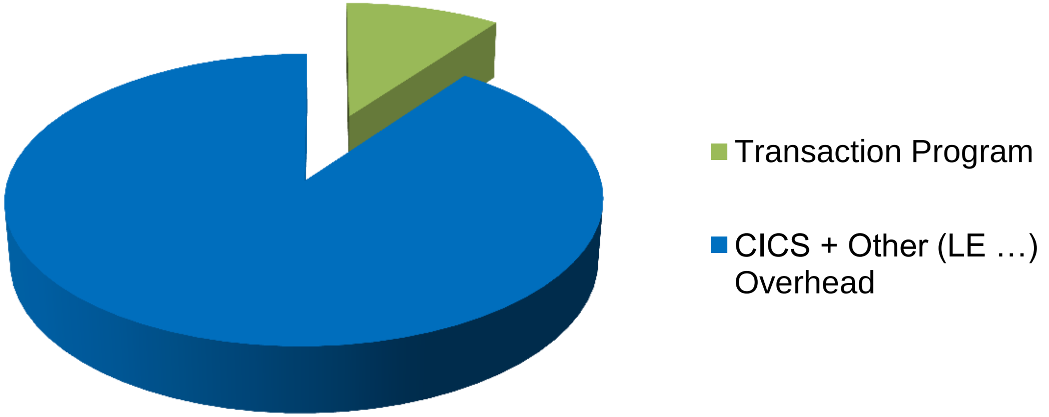
	ORIGINAL DESIGN	CURRENT USAGE
When	Early 70s	Today + Tomorrow
File Organization	ISAM → VSAM	VSAM → DBMS/SQL (DB2)
Need for Data Services	Yes	No (DBMS-Provided)
Connection	BTAM → TN3270	HTTP/S over TCP/IP
Client Device	3270 Workstations	Web-Enabled Devices
User Interface	Green Screens	WUI/GUI → Web Services
Application Training	Yes	No (Self-Service)
Main User Population	Staff + Agents/BP	Clients + Consumers
Total Users	Thousands	Millions
Concurrent Users	Hundreds	Thousands
Transaction Volumes	10s x Thousands	Millions

CICS Overhead Considerations

CICS adds significant overhead to web-initiated traffic which it was not designed for. How much overhead and why?

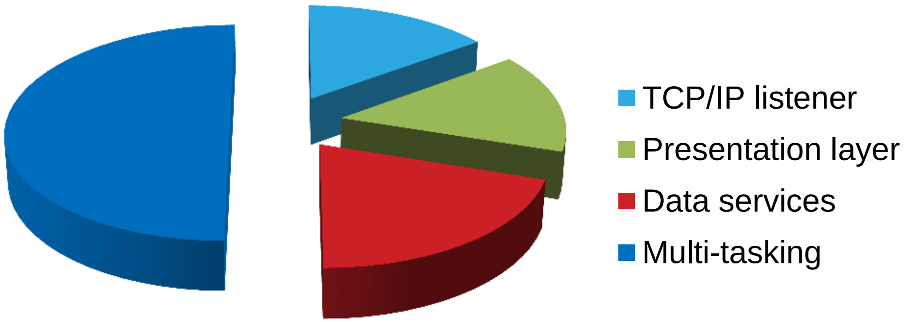
How Much System Overhead?

- Systems (CICS, LE …) = most CPU consumption
- Transaction programs = very little CPU consumption



Where Does CICS Overhead Comes From?

- Event-level TCP/IP listener → costly CICS disruptions
- Presentation layer gets activated by web service calls
- VSAM-intended data services get activated for DBMS
- Not designed for z/OS PE multi-tasking services



Multi-Tasking 101

- MVS multi-tasking:
  - Event Control Blocks (ECB)
  - Pause Elements (PE):
    - Introduced with z/OS
    - Integrated to z/OS lower system layers (control block redesign)
    - IBM → Use PE over ECB for efficiency and performance
    - PE = tool of choice for multitasking subsystems
- CICS multi-tasking:
  - Double-level multitasking: CICS → MVS
  - Relies upon ECB (not IBM-recommended PE)

Transaction Server Prototype

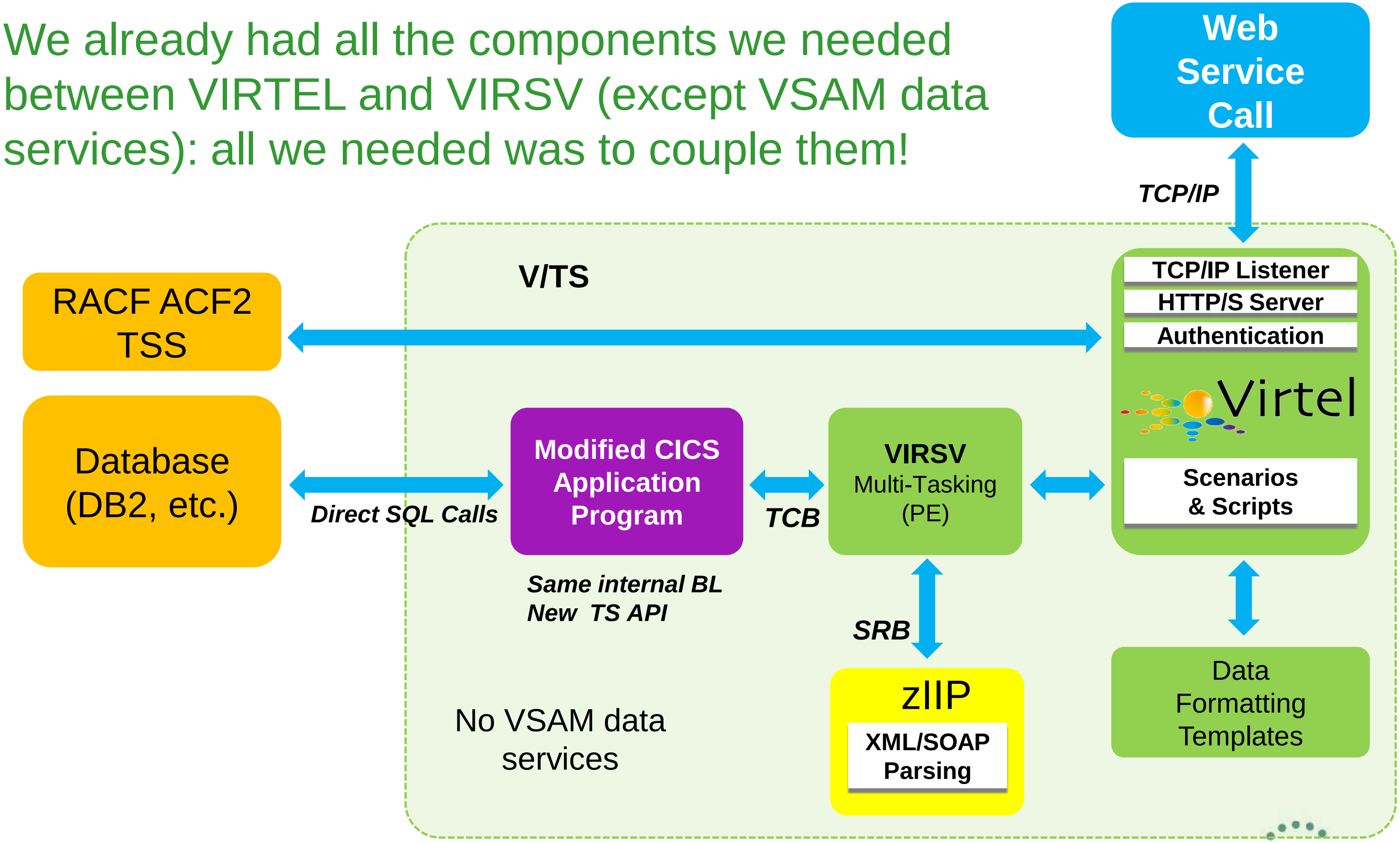
What would a new transaction server specifically designed to run web-initiated CICS transactions with best possible response times, throughput and footprint look like?

Transaction Server Prototype - Design Brief

- Design brief for low-overhead TS prototype:
  - No screen UI: web service only UI
  - Web interface through HTTP/S server with:
    - Message-level (rather than event-level) TCP/IP listener
    - Web service management facility: scripts and scenarios
    - zIIP for parser (XML, SOAP …) and other CPU-intensive tasks
    - RACF server-level (rather than user-level) authentication path
  - Efficient PE-based multi-tasking
  - New API: run existing CICS programs under new TS
  - Direct DBMS/SQL calls + reuse existing DBMS data services
  - VSAM data services

Transaction Server Prototype - Architecture

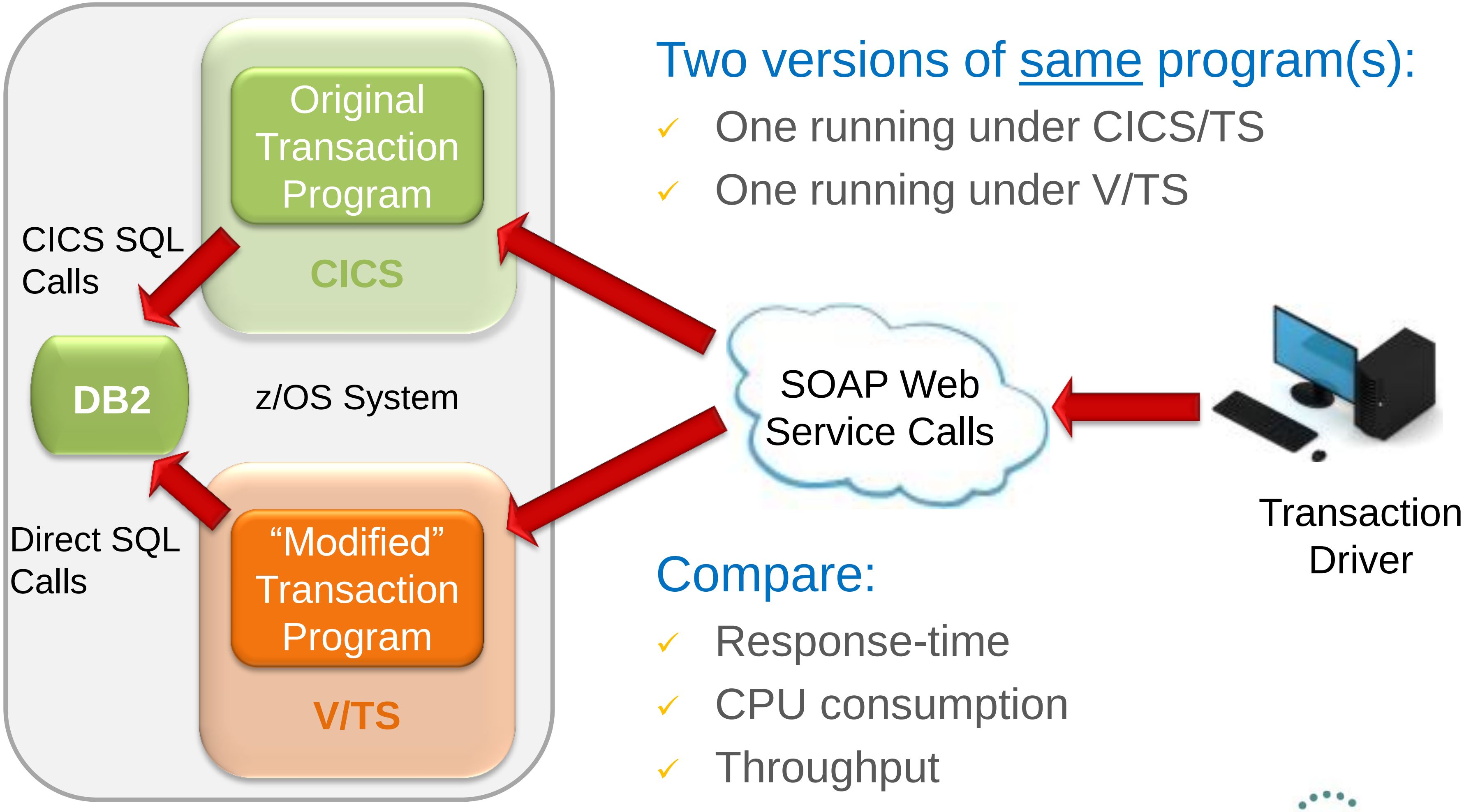
We already had all the components we needed between VIRTEL and VIRSV (except VSAM data services): all we needed was to couple them!



Benchmarking

What kind of response times, throughput and footprint is this new transaction server capable of?

Benchmarking – Methodology and Process



Two versions of same program(s):

- ✓ One running under CICS/TS
- ✓ One running under V/TS

Compare:

- ✓ Response-time
- ✓ CPU consumption
- ✓ Throughput

Scenario #1 – Query Only

- Queries = heaviest (costliest) TP workload = best ROI
- Program used for benchmark:
  - COBOL + DB2
- Business Logic
  - Retrieve user contact data from database
- RACF activation:
  - CICS/TS: first incoming call only (cached authentication)
  - V/TS: each incoming call (no cached authentication)
  - Need to configure Virtel STC to cache RACF control blocks

Scenario #1 - CPU Consumption

QUERY-ONLY	CICS/TS	V/TS	CPU Reduction
TCP/IP	48.71	7.37	84.87%
TP Monitor	427.08	115.32	72.99%
DB9xxxxxxx	25.18	.69	97.26%
RRS	.25	.06	76%
Total	501.22	123.44	75.37%

Scenario #1 - Overall Performance

QUERY-ONLY	CICS/TS	V/TS	
AVG Response Time	327	46	7 times shorter
AVG Throughput	30	152	5.4 times larger
MAX Throughput	35	223	6.4 times larger
CPU Consumption	501	123	4 times smaller

Scenario #2 – All I/O Types

- Updates = 10% or less of TP workload = worse ROI
- Program used for benchmark:
  - COBOL + DB2
- Business Logic
  - Create new user
  - Retrieve new user from database
  - Update new user’s telephone number
  - Update new user’s qualifier
  - Retrieve updated user from database
  - Delete updated user from database
- Work in progress (Not optimized yet)

Scenario #2 - Overall Performance

MIXED I/O TYPES	CICS/TS	V/TS	
AVG Response Time	284	142	2 times shorter
AVG Throughput	35	66	1.9 times larger
MSU Consumption	21465	10194	2.1 times lower

Comparing Scenarios #1 vs. #2 Results

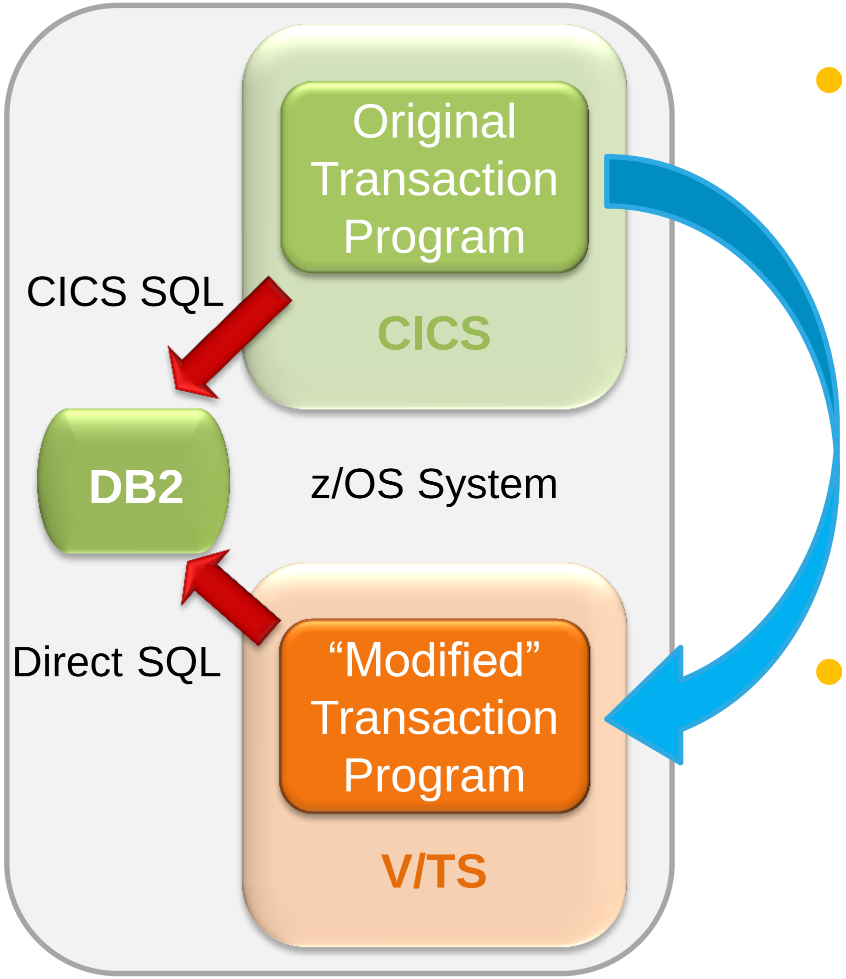
	QUERY-ONLY	MIXED I/Os
AVG Response Time	7 x shorter	2 x shorter
AVG Throughput	5.4 x larger	1.9 x larger
MSU Consumption	4 x lower	2.1 x lower

- **Updates take longer** than queries → TS execution time reduction applies to smaller % of overall response time → overall response time reduction is smaller
- CPU-intensive **DB2 update locking** runs inside TS address space = TS CPU reduction applies to lower % of overall CPU consumption → overall CPU reduction is smaller
- Updates typically only about 10% of all I/O but scenario #2 includes as many updates as queries → worse case scenario
- Scenario #2 not optimized yet (contentions …) → Results might improve
- Current scenario #2 results still = excellent improvement

Program Changes

What kind of changes must be applied to CICS programs to run under V/TS? Could those changes be automated?

Program Changes – Overview



- Interface with transaction server (V/TS vs. CICS):
  - ✓ Parameter list
  - ✓ Processing flow
  - ✓ Initialization section
  - ✓ Core section
  - ✓ Termination section
- File and DBMS accesses:
  - ✓ Direct DB2/SQL calls
  - ✓ [VSAM calls]

Program Changes – Parameter List

V/TS Parameter List:

- ✓ Address of input data area
- ✓ Length of input data area
- ✓ Address of output data area
- ✓ Length of output data area
- ✓ Return code

V/TS

```
      COPY VSVCLIST.
```

```
01   LIST-POINTER.
     03   LIST-ADR-REQU        USAGE   IS POINTER.
     03   LIST-ADR-REQUL       USAGE   IS POINTER.
     03   LIST-ADR-RESP        USAGE   IS POINTER.
     03   LIST-ADR-RESPL       USAGE   IS POINTER.
     03   LIST-ADR-RC          USAGE   IS POINTER.
```

Program Changes – Processing Flow

CICS/TS

```
PROCEDURE DIVISION.

       START-OF-PROGRAM.

   1      PERFORM INITIATE-PROCESS
            THRU INITIATE-PROCESS-XIT.

   2      PERFORM MAIN-PROCESS
            THRU MAIN-PROCESS-XIT.

   3      PERFORM TERMINATE-PROCESS
            THRU TERMINATE-PROCESS-XIT.

          EXEC CICS RETURN
          END-EXEC.
```

Repeat transaction processing:

- ✓ Was handled by CICS
- ✓ Now handled by program

V/TS

```
PROCEDURE DIVISION.

    PERFORM UNTIL NO-MORE-PROCESS

 1       PERFORM INITIATE-PROCESS
           THRU INITIATE-PROCESS-XIT

 2       PERFORM MAIN-PROCESS
           THRU MAIN-PROCESS-XIT

 3       PERFORM TERMINATE-PROCESS
           THRU TERMINATE-PROCESS-XIT

           END-PERFORM.

       Z000-EXIT.
            GOBACK.
```

1 Initialization 2 Core Processing 3 Termination

Program Changes - Initialization Section

CICS/TS

```
INITIATE-PROCESS.

     EXEC CICS ADDRESS EIB(DFHEIBLK)
         END-EXEC.
     MOVE SPACES TO WORK-DFHCOMMAREA.
     MOVE EIBCALEN TO EIBCALEN-DECIMAL.
     IF EIBCALEN-DECIMAL > MAX-COMMAREA
         MOVE RC-TOO-LONG-COMMAREA TO WORK-RETURN-CODE
         MOVE MSG-TOO-LONG-COMMAREA TO
         WORK-ERROR-MESSAGE
         GO TO INITIATE-PROCESS-XIT
     ELSE
         MOVE DFHCOMMAREA TO WORK-DFHCOMMAREA.
     MOVE 0 TO WORK-RETURN-CODE.
     MOVE SPACES TO WORK-ERROR-MESSAGE.

INITIATE-PROCESS-XIT.
     EXIT.
```

Copy input data:

- ✓ From COMMAREA
- ✓ To local working storage area

V/TS

```
2000-CALL-VSVPSYNC SECTION.

     CALL 'VSVPSYNC' USING REFERENCE POINTER-OF-POINTER
                     RETURNING CODE-PSYNC-NUM
     IF CODE-PSYNC-NUM NOT ZERO
         GO TO 2099-FIN
     END-IF.

 2030-SET-VIRSV-POINTERS.

     SET ADDRESS OF LIST-POINTER  TO POINTER-OF-POINTER
     SET ADDRESS OF REQU          TO LIST-ADR-REQU
     SET ADDRESS OF REQUL         TO LIST-ADR-REQUL
     SET ADDRESS OF RC            TO LIST-ADR-RC.

 2030-SET-SCENARIO-POINTERS.

     SET ADDRESS OF VTSADD-ACTION    TO REQU-ACTION-PTR
     MOVE VTSADD-ACTION TO WORK-ACTION
     SET ADDRESS OF VTSADD-COMMAND   TO REQU-COMMAND-PTR
     MOVE VTSADD-COMMAND TO WORK-COMMAND
     SET ADDRESS OF VTSADD-DATA-WORK-AREA
                 TO REQU-DATA-WORKAREA-PTR
     SET ADDRESS OF VTSADD-RETURN-CODE TO REQU-RETCODE-PTR
     SET ADDRESS OF VTSADD-ERROR-MESSAGE TO REQU-ERROR-MSG-PTR.

 2099-FIN.
     EXIT.
```

Program Changes – Core Section

CICS/TS

```
B110-RECEIVE-MAP.

    PERFORM B111-DO-THE-DB2-WORK
       THRU B111-DO-THE-DB2-WORK-EXIT.
    MOVE WORK-DFHCOMMAREA TO VTSCOMM.
    MOVE DATA-WORK-AREA TO PRESET-DWA.
    EXEC CICS WRITEQ TS
        QUEUE (PRESET-OPTIONS-QNAME)
        FROM (PRESET-OPTIONS-QUEUE)
        LENGTH (PRESET-OPTIONS-LENGTH)
        ITEM (PRESET-OPTIONS-ITEM)
        MAIN
        END-EXEC.

B110-RECEIVE-MAP-EXIT.
    EXIT.
```

V/TS

```
 B110-RECEIVE-MAP.

    PERFORM B111-DO-THE-DB2-WORK
       THRU B111-DO-THE-DB2-WORK-EXIT.
    MOVE WORK-DFHCOMMAREA TO VTSCOMM.
    IF VIRSV-PROCESS
        MOVE DATA-WORK-AREA TO VTSDWA
        GO TO B110-RECEIVE-MAP-EXIT.
    MOVE DATA-WORK-AREA TO PRESET-DWA.

B110-RECEIVE-MAP-EXIT.
    EXIT.
```

Simple EXEC CICS replacement:

- ✓ Replace Temporary Storage Queue logic with equivalent V/TS logic
- ✓ LINK to V/TS through dynamic calls

Program Changes – Termination Section

CICS/TS

```
TERMINATE-PROCESS.

    MOVE WORK-COMMAREA TO DFHCOMMAREA.

TERMINATE-PROCESS-XIT.
    EXIT.
```

V/TS

```
TERMINATE-PROCESS.

    MOVE DATA-WORK-AREA TO VTSDWA.
    MOVE WORK-RETURN-CODE TO VTSRC.
    MOVE WORK-ERROR-MESSAGE TO VTSMSG.

TERMINATE-PROCESS-XIT.
    EXIT.
```

Copy output data:

- ✓ From local working storage area
- ✓ To COMMAREA

Program Changes – DBMS Accesses

CICS/TS

```
EXECUTE-DB2-FILE.

    EXEC SQL PREPARE DYNAMSELECT
        FROM :COMMAND-STATEMENT
        END-EXEC.
    PERFORM PROCESS-ERROR-CODE
       THRU PROCESS-ERROR-CODE-XIT.
    IF HBWORK-RC-SUCCESS
        NEXT SENTENCE
    ELSE
        GO TO EXECUTE-DB2-FILE-XIT.
    EXEC SQL EXECUTE DYNAMSELECT
        USING :HOLD-WORK-AREA-ACTUAL
        END-EXEC.

EXECUTE-DB2-FILE-XIT.
    EXIT.
```

V/TS

```
EXECUTE-DB2-FILE.

    EXEC SQL PREPARE DYAMSELECT
        FROM :COMMAND-STATEMENT
        END-EXEC.
    PERFORM PROCESS-ERROR-CODE
       THRU PROCESS-ERROR-CODE-XIT.
    IF HBWORK-RC-SUCCESS
        NEXT SENTENCE
    ELSE
        GO TO EXECUTE-DB2-FILE-XIT.
    EXEC SQL EXECUTE DYNAMSELECT
        USING :HOLD-WORK-AREA-ACTUAL
        END-EXEC.

EXECUTE-DB2-FILE-XIT.
    EXIT.
```

Direct access to DB2/SQL:

- ✓ No change to application code
- ✓ Use IBM standard DB2/SQL protocol and data services
- ✓ Avoid redundant CICS data services

Program Changes – Automation Potential

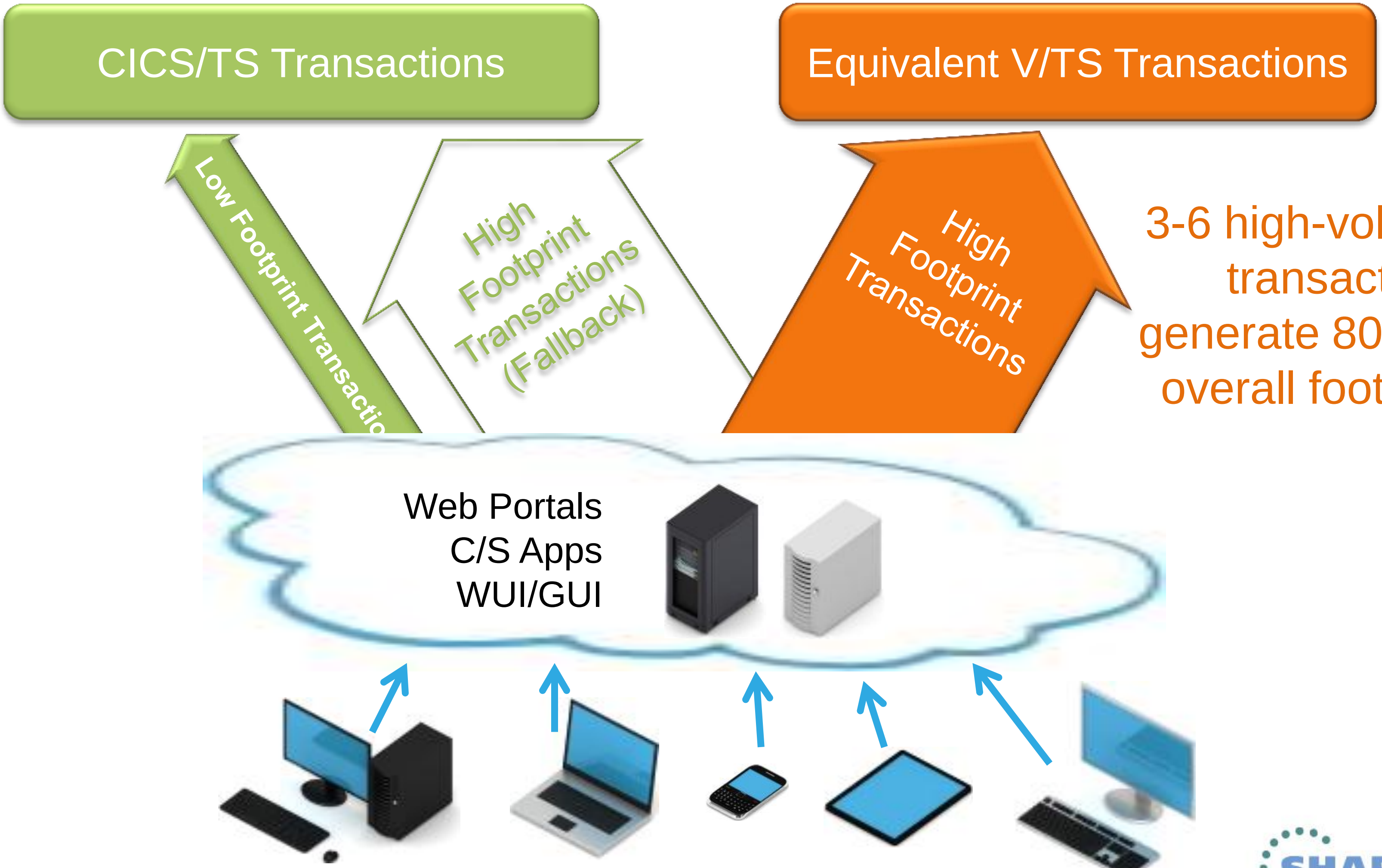
- Initial onetime code modification:
  - Manual at start → develop some experience
  - Partly automated later on?
- Techniques to avoid dual maintenance:
  - Automated conversion tools → pre-compilation
  - Hide code differences behind copybooks/includes
  - Insert dual logic for execution-time
- Most maintenance changes will fall <u>outside</u> TS interface

Implementation Strategy

Suggested Implementation

Key Implementation Considerations

Suggested Implementation


CICS/TS Transactions
Equivalent V/TS Transactions
Low Footprint Transactio
High Footprint Transactions (Fallback)
High Footprint Transactions
Web Portals
C/S Apps
WUI/GUI

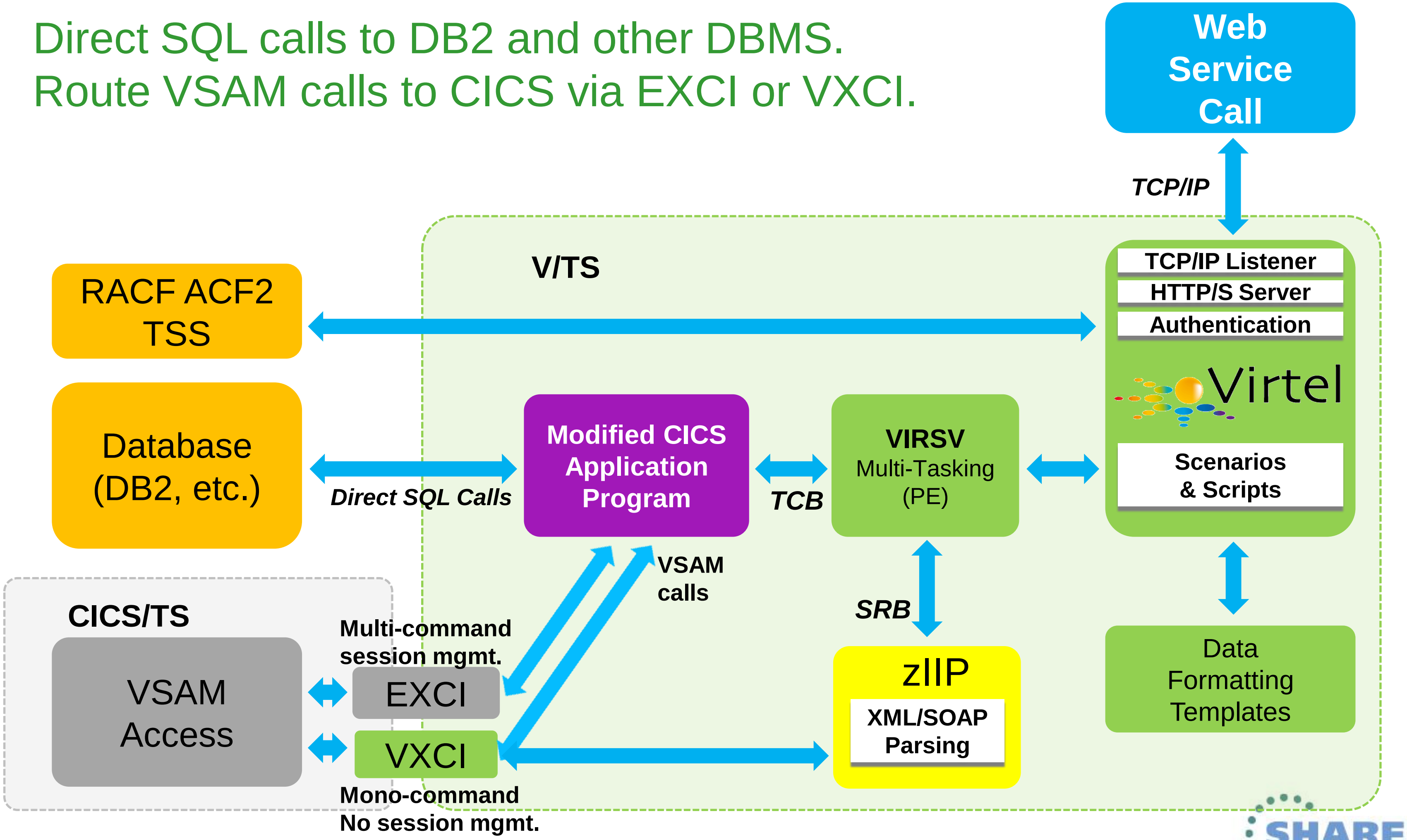
3-6 high-volume transactions generate 80% of overall footprint

Key Implementation Considerations

- Limited scope
  - 3-6 transactions generate 80% of web-initiated transaction volume
  - Don’t replace or eliminate CICS: create alternate path
- Risk management
  - Retain CICS transaction path for fallback and (debug) reference
- Avoid dual maintenance
  - Automate code changes with pre-compiler
  - Hide changes that cannot be automated behind copy/includes
  - Most maintenance changes will fall outside TS interface
- Real world = need VSAM support
  - Currently: DBMS/SQL only (DB2, etc)
  - Short term: use EXCI or VXCI to submit VSAM I/Os through CICS
  - Long term: develop “minimum” VSAM support?

VSAM Support – “Hybrid” Solution

Direct SQL calls to DB2 and other DBMS.
Route VSAM calls to CICS via EXCI or VXCI.



Cost/Benefit Analysis

Mainframe organizations are in cost containment mode!

−

APPLICATIONS / SYSTEMS:

- ✓ Unfamiliar technology & skillset
- ✓ Untested and untrusted
- ✓ Code changes
- ✓ Duplicate maintenance
- ✓ Debugging

+

CFO / CIO / OPS MGR:

- ✓ Smaller footprint → reduced MF costs
- ✓ Shorter response times → customer satisfaction
- ✓ Increased throughput → delayed HW upgrade + growth support

Conclusions and Next Steps

What have we proven and where do we go from here?

Conclusions

- Verified initial overhead assumption:
  - 90-99% of transaction footprint is system (CICS, LE, etc) overhead
- Identified alternative to greatly improve:

	Query-Only	Updates
Response times:	7 times faster	Twice faster
Throughput:	6 times larger	Twice larger
Footprint:	4 times smaller	Twice smaller

- Expected annual cost savings = $100Ks to $millions
- Main issues:
  - Risk: unfamiliar + untested + untrusted
  - Code modifications + long-term support
  - No VSAM support to date → Limited “real life” applicability
- Cost/benefits analysis

Next Steps

- Confirm DB2/SQL benchmark results with real customer transaction in real production environment
- Add VSAM support
- Then …

The best way to predict the future is to create it

Abraham Lincoln

Questions

SHARE expo booth # 521 (VIRTEL)

patrick.fournier@syspertec.us

+1 (925) 954-9649