SQL Server Notifications in a manufacturing environment

David Johnson

www.djohn89.com

The views expressed in this presentation are my own opinions and do not necessarily correspond to endorsements or opinions of Fiat Chrysler Automobiles (FCA) or Mobis North America (MNA).

Outline

- Introduction to Query Notifications
  - Example: software bug tracking
  - Polling, Triggers, and Notifications
  - Message Queues
- Case Study: Manufacturing Execution System (MES)
  - Assembly line stations, torque guns, barcode scanners, RFID readers, and testing equipment
  - Industrial PCs assist operators and enforce rules
  - Examples: Polling for vehicles, test results, and problematic incoming orders
- Conclusions

Introduction to Query Notifications

- Query Notifications are a caching mechanism intended to alleviate repetitious queries of data that do not change frequently
- Rather than polling a table and looking for changes:
  - Register a SQL query indicating what you want to know
  - Wait for SQL Server to send you a message indicating that the table has changed*
  - Perform your query or action
- *You may receive notifications whenever the server can’t guarantee that your cache is still valid

Example: software bug tracking

- Suppose that you use a bug tracking program which has a table like this:

	DefectID	Created	LastModified	Summary	Severity	Status	AssignedToUserID	CreatedByUserID	ProjectID
1	1	2013-05-...	2013-05-23...	MP3 files crash system	3	1	1	2	1
2	2	2013-05-...	2013-05-09...	Text is too big	0	4	NULL	3	1
3	3	2013-05-...	2013-05-19...	Sky is wrong shade of blue	1	2	4	5	2
4	4	2013-05-...	2013-05-23...	Can't play files more than 200 bytes long	2	3	1	1	1
5	5	2013-05-...	2013-05-15...	Installation is slow	0	2	2	2	1
6	6	2013-05-...	2013-05-29...	DivX is choppy on Pentium 100	2	1	1	6	1
7	7	2013-05-...	2013-05-10...	Client acts as virus	3	4	NULL	3	2
8	8	2013-05-...	2013-05-23...	Subtitles only work in Welsh	2	2	2	1	1
9	9	2013-05-...	2013-05-15...	Voice recognition is confused by background noise	1	4	NULL	5	2
10	10	2013-05-...	2013-05-09...	User interface should be more caramelly	0	0	1	2	2
11	11	2013-05-...	2013-05-29...	Burning a CD makes the printer catch fire	3	4	NULL	6	1

- Your manager, Greg, says that everyone should receive an email within 1 minute whenever there are at least 3 bugs, unclosed (Status=4) with a Severity of 3.
- How do you fulfill Greg's request?

SQL Query for severe, unclosed bugs

```
SELECT *  FROM [dbo].[Defect]
  where [Severity] = 3 and [Status] = 4
```

	DefectID	Created	LastModified	Summary	Severity	Status	AssignedToUserID	CreatedByUserID	ProjectID
1	7	2013-05-08 00:00:00.000	2013-05-10 00:00:00.000	Client acts as virus	3	4	NULL	3	2
2	11	2013-05-10 00:00:00.000	2013-05-29 00:00:00.000	Burning a CD makes the printer catch fire	3	4	NULL	6	1
3	17	2013-05-12 00:00:00.000	2013-05-13 00:00:00.000	Password displayed in plain text	3	4	NULL	1	2
4	26	2013-05-16 00:00:00.000	2013-05-18 00:00:00.000	Password reset changes passwords for all users	3	4	NULL	6	2

- At the moment, there are 4 bugs in the Defect table matching these criteria.
- Greg tells you to mark DefectIDs 7 and 11 as invalid (Status=2).
- But how will you know when another severe bug gets entered into this table?
- Possible solutions: polling, triggers, and notifications

Solution 1: Polling

- You edit one of your in house applications to repeat that SQL query **every minute** and send an email whenever the criteria are met
- Good: simple, deterministic, and fast (on your development system). Works with any db.
- Bad: Polling from multiple computers increases the database load, which slows down all SQL queries.
- Ugly: What if an update statement changes two rows at the same time in between polls?

Polling Code Example

```
private int lastCount = 0;
private void timer1_Tick(object sender, EventArgs e)
{
  using (SqlConnection connection = new SqlConnection())
  {
    using (SqlCommand command =
      new SqlCommand("SELECT count([DefectID]) C FROM [dbo].[Defect]
            where [Severity] = 3 and [Status] = 4;", connection))
    {
      connection.Open();
      using (SqlDataReader reader = command.ExecuteReader())
      {
        if (reader.Read())
        {
          int newCount = reader.GetInt32(0);
          if (newCount != lastCount && newCount >= 3)
          {
            lastCount = newCount;
            SendEmail();
          }}}}}}
```

- Timer1 is a Windows Forms Timer
- Still has problems: how frequently to poll? How reliable is it?

Multiple simultaneous updates

Before updates:

	DefectID	Created	LastModified	Summary	Severity	Status	AssignedToUserID
1	1	2013-05-...	2013-05-23...	MP3 files crash system	3	1	1
2	17	2013-05-...	2013-05-13...	Password displayed in plain text	3	4	NULL
3	26	2013-05-...	2013-05-18...	Password reset changes passwords for all users	3	4	NULL

```
UPDATE [dbo].[Defect]
        SET [Severity] = 3, [Status] = 4
        WHERE [DefectID] = 1

UPDATE [dbo].[Defect]
        SET [Severity] = 1, [Status] = 1
        WHERE [DefectID] = 26
```

After updates:

	DefectID	Created	LastModified	Summary	Severity	Status	AssignedToUserID
1	1	2013-05-01 ...	2013-05-23 ...	MP3 files crash system	3	4	1
2	17	2013-05-12 ...	2013-05-13 ...	Password displayed in plain text	3	4	NULL
3	26	2013-05-16 ...	2013-05-18 ...	Password reset changes pass...	1	1	NULL

- If these two updates happen between polls (or inside a transaction), no email is sent
- Otherwise, an email will be sent. Possibly unreliable.

Polling - Summary

- Simple but inefficient, polling is the traditional model for getting updates
- The polling interval is always a problem
- Timers can be unreliable in Windows

Solution 2: Triggers

- You write a database triggers for AFTER INSERT and AFTER UPDATE. The triggers redo the SQL query and send emails using dbmail.
- Good: The trigger only runs once, and it runs only on the database server, so only 1 email gets sent.
- Bad: The trigger always runs, even when unrelated rows are changed. The database is slower than it should be.
- Ugly: You accidentally introduce a bug in the trigger and it silently corrupts your data for two days before anyone notices!

Insert Trigger TSQL Code

```
CREATE TRIGGER dbo.tri_i_Defect  ON  dbo.Defect
AFTER INSERT AS BEGIN
SET NOCOUNT ON;

declare @newCount int, @addedCount int;

SELECT @newCount = count([DefectID])
FROM [dbo].[Defect]
where [Severity] = 3 and [Status] = 4;

SELECT @addedCount = count([DefectID])
from inserted
where [Severity] = 3 and [Status] = 4;

if(@addedCount > 0 and @newCount >= 3)
            exec msdb.dbo.sp_send_dbmail;
END
```

- Also need to write similar triggers for update and delete statements
- Be very careful about error handling!

Originating Statement

```
INSERT INTO [dbo].[Defect]
        ([Created],[LastModified],[Summary],[Severity],[Status]
        ,[AssignedToUserID],[CreatedByUserID],[ProjectID])
    VALUES (getdate(),null,'Changing fonts causes all text to reverse (right to left)',4,1
        ,null,2,1)
```

↳ **TRIGGER dbo.tri_i_Defect runs**

↳ Email might be sent

Transaction commits

- The triggers run in the context of the originating statement (i.e., the transaction)
- If trigger fails, then the originating statement fails! (Transaction aborts.)
- The trigger always runs, even when it's irrelevant to the business goals.

Triggers - Summary

- Use with caution. Debugging is not easy.
- Triggers are powerful and reliable because they always run! They must be fast because they are integral to the server.
- Triggers run in the context of the originating statement, which can cause seemingly unrelated applications to unexpectedly fail (table locks, cascading triggers).

Solution 3: Query Notifications

- You decide to try Query Notifications with C# or VB SqlDependency to receive a callback when the criteria are satisfied.
- Good: reliable, fast, and no redundant queries occur. Data can't be corrupted or lost.
- Bad: You have to learn a new way of doing database queries.
- Ugly: It's Microsoft specific (2005+), but Oracle has a similar mechanism

Query Notification Example

```
private string DbConnectionString = "Server=some-server.mycompany.com;
          Initial Catalog=defects; Integrated Security=True";
private SqlConnection BrokerConnection = null;
private SqlCommand BrokerCommand = null;
private DataSet BrokerDataSet = null;

public void CreateDep()
{
  SqlDependency.Start(DbConnectionString);

  BrokerConnection = new SqlConnection(DbConnectionString);
  BrokerCommand = new SqlCommand("SELECT [DefectID] FROM [dbo].[Defect]
          where [Severity] = 3 and [Status] = 4;", BrokerConnection);
  BrokerDataSet = new DataSet();

  StartListening();
}
```

- Part 1: Initialize the .NET runtime's background thread that manages all QNs in the application

QN Example – Part 2

```
private void StartListening()
{
  // Make sure the command object does not already have
  // a notification object associated with it, and there aren't any old results
  BrokerDataSet.Clear();
  BrokerCommand.Notification = null;

  // Create and bind the SqlDependency object to the command object.
  SqlDependency dependency = new SqlDependency(BrokerCommand);
  dependency.OnChange += new OnChangeEventHandler(OnDataChanged);

  // we don't have to care about these results,
  // but we must run the query at least once to receive notifications
  using (SqlDataAdapter adapter = new SqlDataAdapter(BrokerCommand))
    adapter.Fill(BrokerDataSet, "Defect");
}
```

- Part 2: Create the SqlDependency and tell SQL server to start listening for changes

QN Example – Part 3

```
void OnDataChanged(object sender,
        SqlNotificationEventArgs e)
{
  // Check COUNT(*) from table, then possibly send email
  // This event will occur on the worker thread pool.
  SendEmail();

  // must redo the query to resubscribe
  StartListening();
}
```

- Part 3: Receive a callback from the worker thread.
- Optional: subscribe again

QN Example – Part 4

```
private void Form1_FormClosed(object sender,
FormClosedEventArgs e)
{
  RemoveDep();
}

public void RemoveDep()
{
  // Release the dependency.
  SqlDependency.Stop(DbConnectionString);

  if (BrokerConnection != null)
    BrokerConnection.Close();
}
```

- Part 4: Clean up. Stop background thread, free server resources.

QN Benefits

```
private SqlCommand BrokerCommand = new SqlCommand(
        "SELECT [DefectID] FROM [dbo].[Defect] where  [Severity] = 3 and
                [Status] = 4;", BrokerConnection);
```

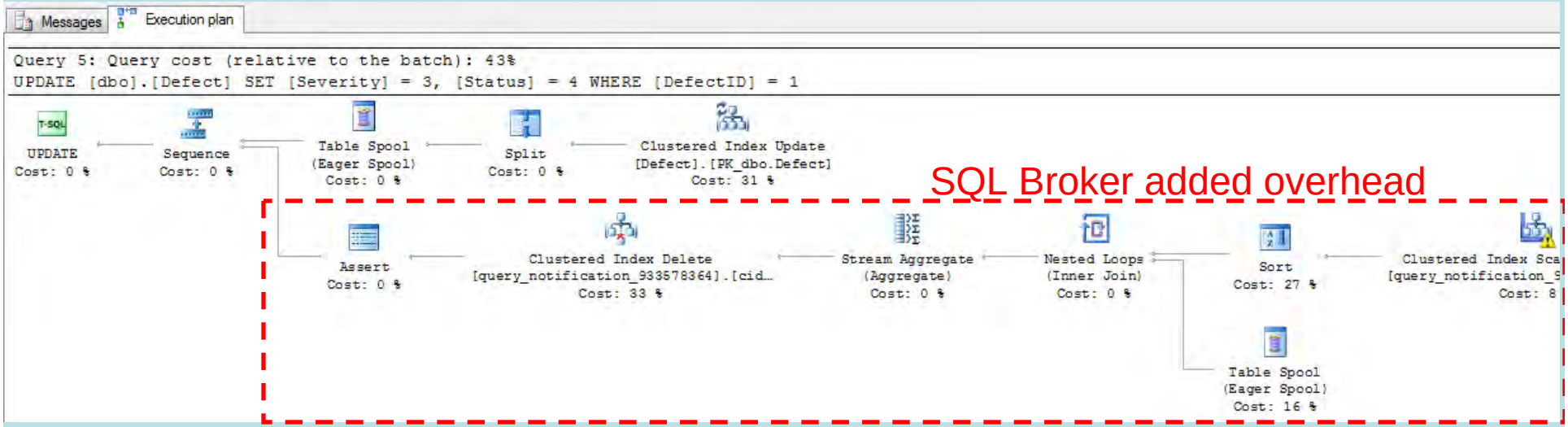
- The same notification works for Insert, Update, and Delete statements
- The notification is guaranteed to be delivered by the SQL Broker
- SQL Server won’t fire the notification unless the WHERE clause is satisfied
- Still need to check COUNT(*) when notification occurs

SQL Broker Operations

- The Broker dispatches a message to your application using a background thread
- Messages are based on meta information from your cached query and the changes to the table
- You could get unwanted notifications if:
  - the server restarts
  - the QN subscription expires
  - ALTER TABLE, DROP TABLE, etc.
  - there are too many simultaneous updates occuring to determine which of them might invalidate your cache

Consequences for other SQL

```
UPDATE [dbo].[Defect]
    SET [Severity] = 3, [Status] = 4
    WHERE [DefectID] = 1
```



- There is a slight cost to other SQL statements (ins/upd/del, not select)
- The cost is similar to updating a non-clustered index (until the QN is removed)

Query Notifications - Summary

- Estimated benefits of QN as compared to polling with 10 db updates per day:
  - Polling: 1 query/second * 86,400 sec/day * 100 clients = 8.6 M select queries.
  - QN: 100 clients * 3 queries = 300 select queries
    - (1: setup, 2: callback, 3: resubscribe)
- Query Notifications are easiest in C# or VB.NET with SqlDependency, but they are possible in any ODBC client
- Very reliable, but slightly slower than triggers

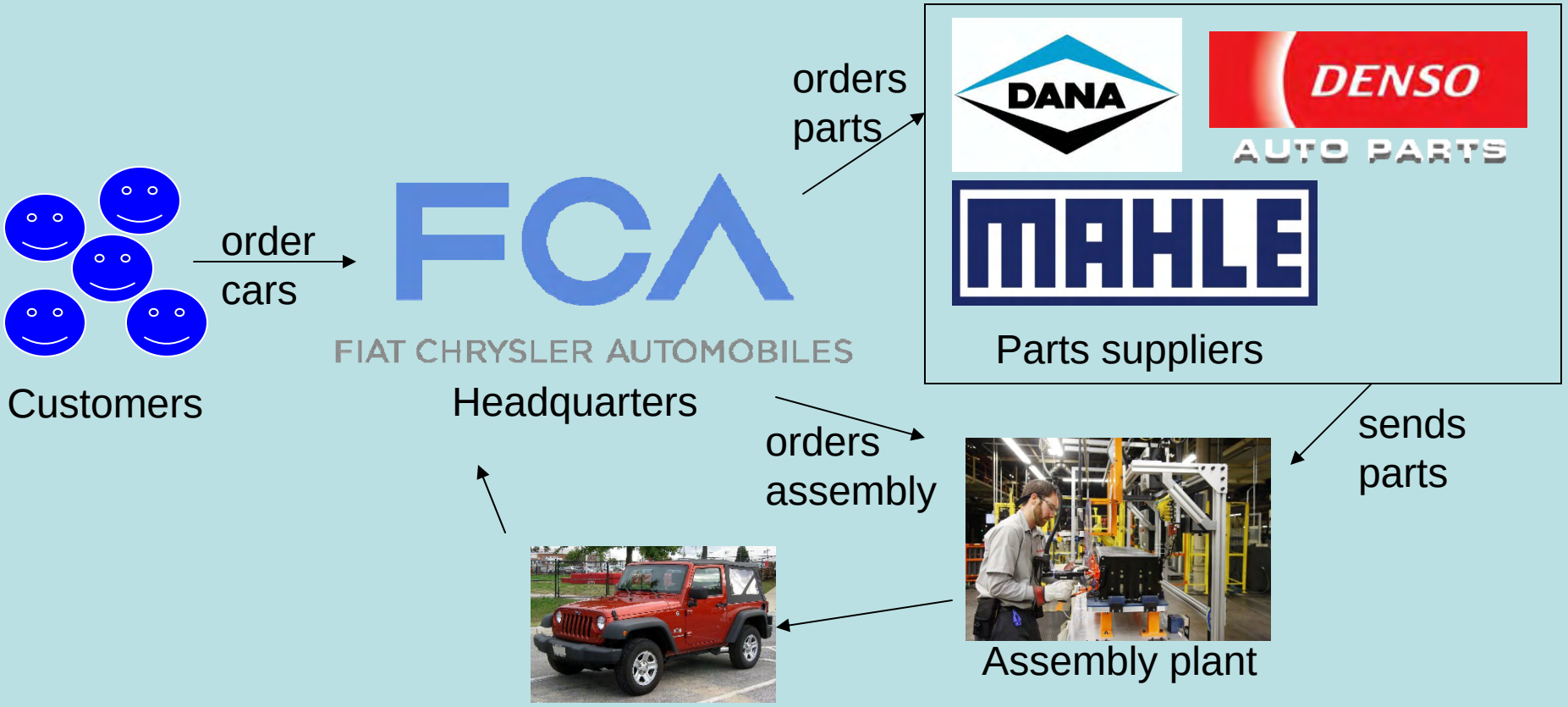
Message Queues

- SQL Server 2005+ has a message queuing service (Broker)
  - Used by Query Notifications, Database email, external clients
- Scales up to millions of messages per minute
- Message management is transactional, not protocol oriented

Transactional message mgmt.

- SQL Server makes message management easy because it uses transactions
  - If the client fails, the transaction aborts using the standard database mechanisms
- Unlike IBM Websphere MQ, the receiving client can be remote
- Unlike Apache ActiveMQ, queues can be bidirectional

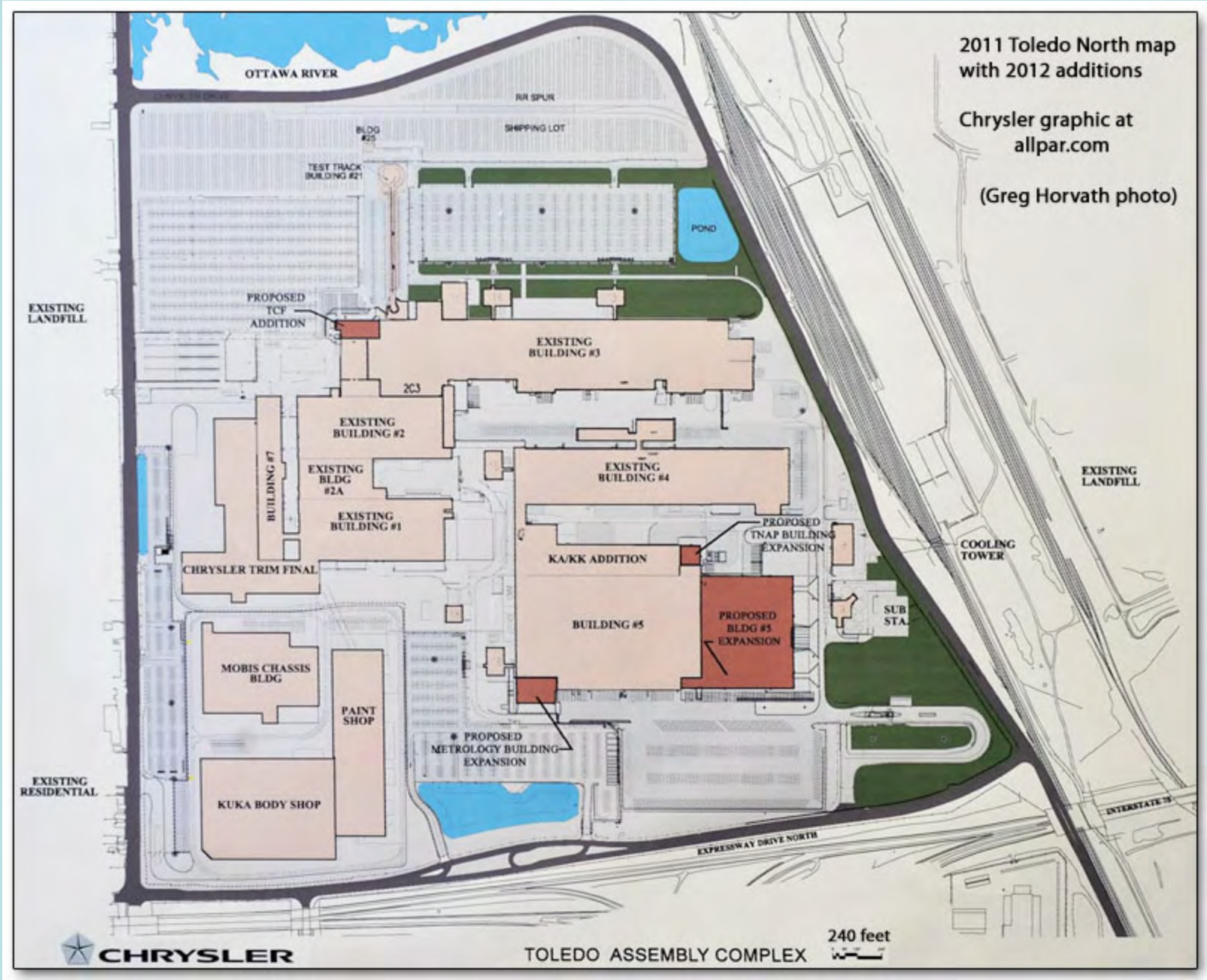
Uses of a Message Queue



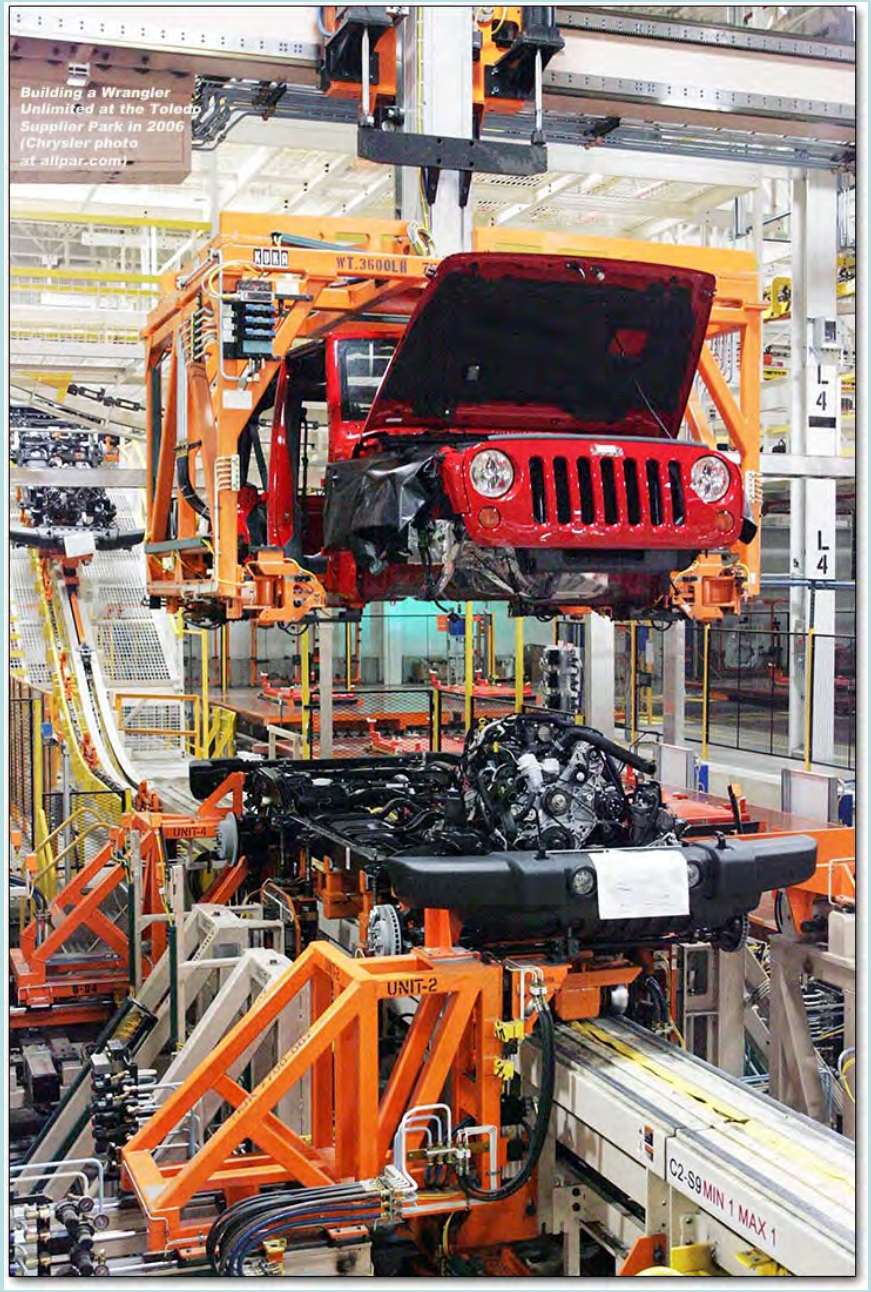
- Each stage has at least one message queue
- Queue allows either sending or receiving computers to crash without loss of information
- Much like email, but for programs not people

Questions?

- Next: Case Study in an automotive assembly plant

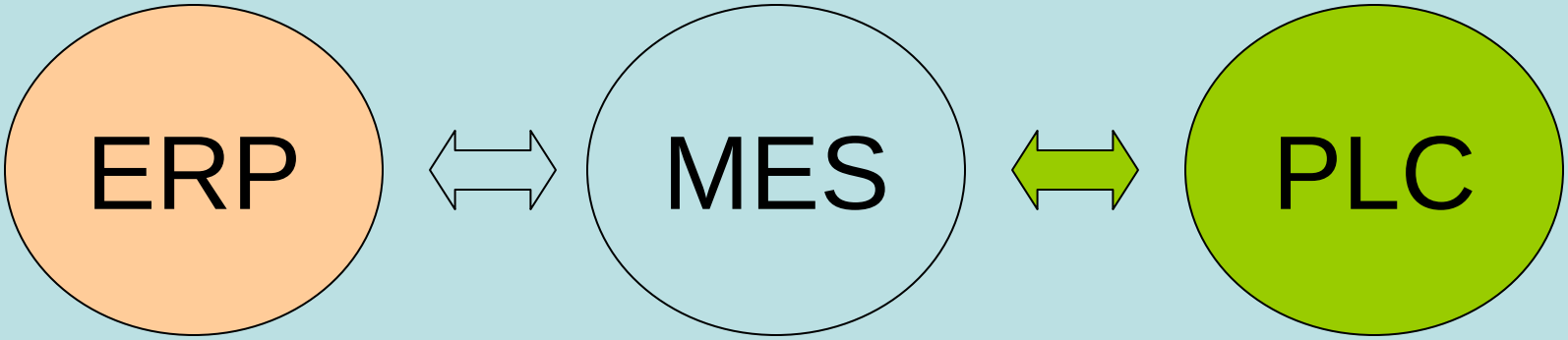


www.allpar.com



www.allpar.com

Assembly Plant Computer Systems



Enterprise Resource Planning

- Incoming orders
- Inventory management
- Billing
- Reports & Auditing

Manufacturing Execution System

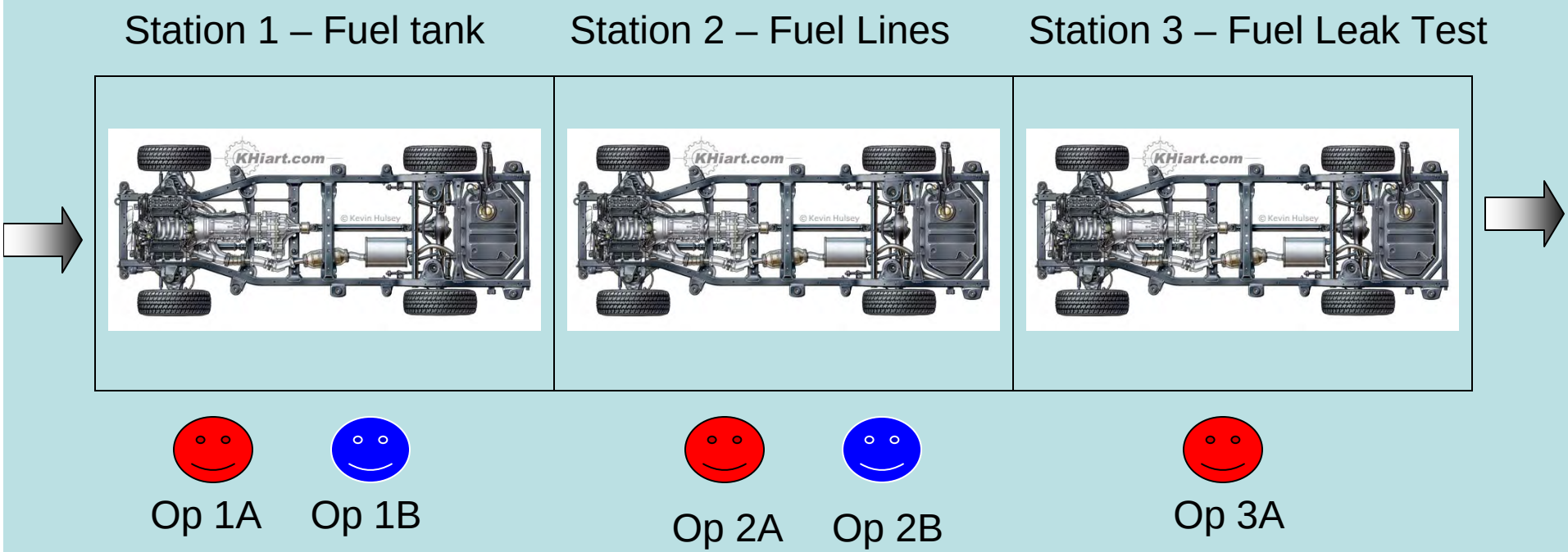
- Production database
- Individual computers (workstations, testing systems, inspections)
- Barcode scanners
- Torque guns

Programmable Logic Controllers

- Motors
- Sensors
- Robots
- Lights
- Buttons

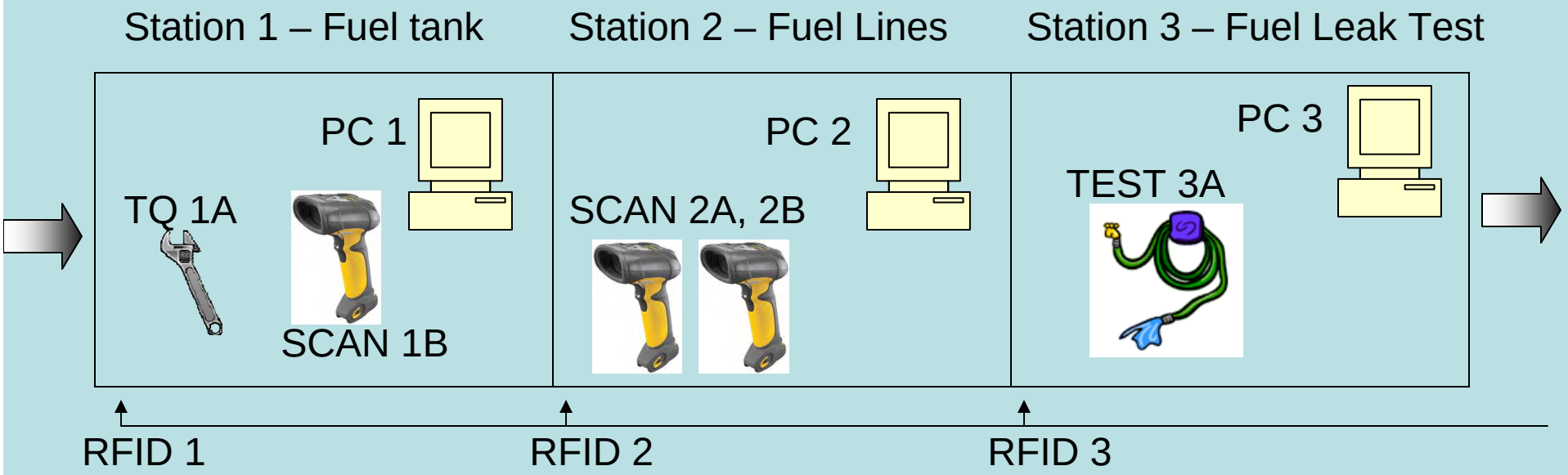
Each of these types of computer systems performs a different role in automotive assembly.

MES Intro – Fictional Assembly Line



- Operators at each station perform assembly tasks
  - E.g., Operator 1A attaches a fuel tank with two bolts. Op 1B scans a barcode on the fuel tank
  - Op 2A and 2B attach fuel line hoses and scan barcodes
  - Op 3A uses a leak testing system to verify that the fuel system is correctly installed

Intro – Assembly Line Computers



- RFID readers determine which vehicle is in which station
- Industrial PCs display instructions to the operators and control the other devices based on the current vehicle
- Torque wrenches are used to attach parts
- Barcode scanners are used to verify part numbers

Intro – Torque Guns

Manual torque wrench (“clicker”)

Electric nutrunner (“torque gun”)

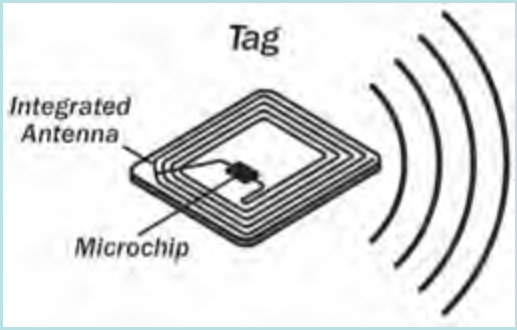
- Manual torque wrenches (“clickers”) are too slow and inaccurate
- Electronic torque guns are fast and precise. Operator fatigue and injury are also reduced.
  - Industrial PC tells the torque guns: how many torques on each part (e.g., 4 bolts), how much torque to apply (50 Nm), how many times to rotate (350°-450°)
  - Torque gun responds with data about what the operators actually did

Intro – Barcode Scanners

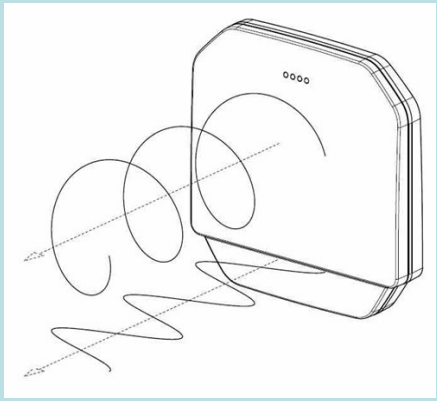
Rear axle barcodes

- The industrial PC enforces the following rules:
- After attaching a part, the part number must be scanned to make sure the right part was used
- The serial number must be scanned for billing
- Part numbers are often 8 digits followed by a 2 letter revision level (e.g., 12345678AB)
- Prefixes or checksums can be used to validate these inputs

Intro – RFID Tags and Readers

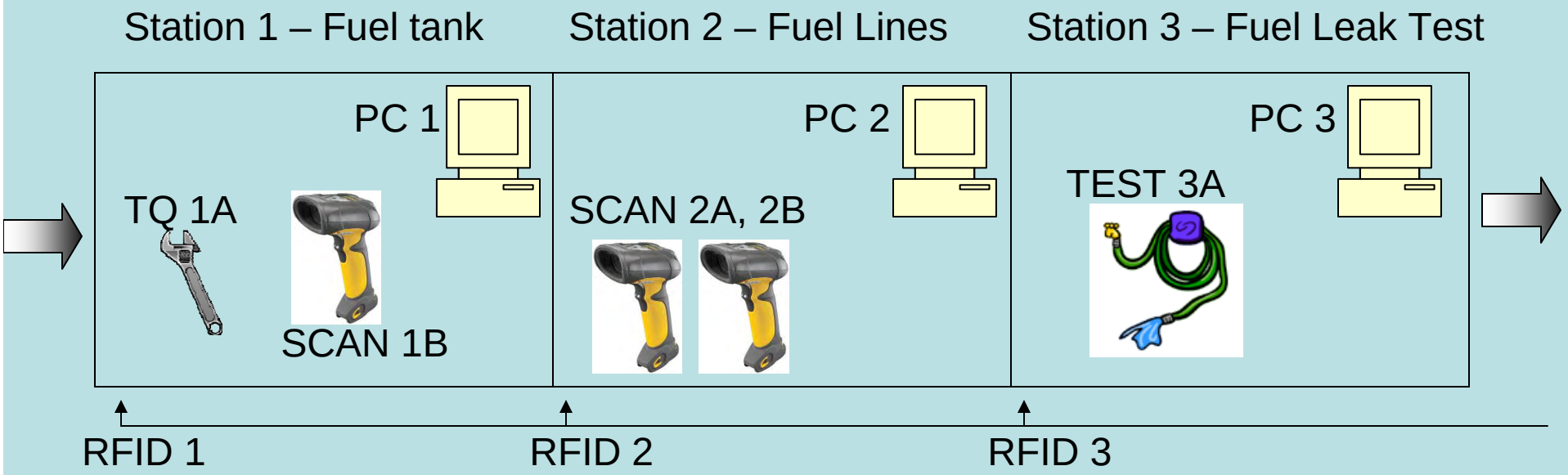


Passive RFID tag


RFID Reader

- The integrity of the RFID reader system is critical because the vehicles move through many different stations
- The Industrial PC reads database records to determine which vehicle is in which station
- Other database queries determine the operations and parts needed at the station for the current vehicle

Review of MES



- Industrial PCs display instructions to the operators and control the torque guns, barcode scanners, and other devices based on the current vehicle
- The operators rely on the Industrial PCs
  - To tell them which operations to perform
  - To enforce quality rules
  - To update the MES database server

Problem 1: Polling for vehicles

- How does each Industrial PC know that a new vehicle has arrived in station?
  - The RFID reader program updates a database table when it reads new tags
  - Traditional approach: poll the RFID table
  - Alternative approach: use query notifications
- Polling is the normal computer architecture for industrial line operations

PLCs are always in charge

Siemens Simatic PLC: a real time industrial control system for hundreds of devices, tolerant of extreme variations in power, temperature, vibration, electronic noise, and physical impact.

- Programmable Logic Controllers (PLCs) often use polling in normal operations:
- (ladder logic)

```
while(true)
        if(limit switch 1 is true and torque gun 5 has been used more than 4 times)
                turn on alarm 32;
        sleep(100);
```

- Polling is OK for PLCs because they use dedicated hardware on realtime systems
- But polling is a terrible idea MES computers

Polling for vehicles – RFIDTable

Tag	VIN	Station	Timestamp
50034	1C4	1	08:30:27
50057	1C5	2	08:31:53
30029	1C6	3	08:29:38

Suppose PC 2 wants to know when a new vehicle is in station 2:

```
OldVIN = ‘’
NewVIN = ‘’
While true:
        SELECT @NewVIN=VIN from RFIDTable where Station=2
        If NewVIN != OldVIN
                OldVIN = NewVIN
                Do stuff
        Sleep(1000)
```

Alternative: Query Notifications

- Each Industrial PC registers a QN for one row in the database table (by Station)
- When the RF-ID reader writes a new tag, the Industrial PC receives the notification and provides new instructions to the operators (part numbers, quality checks, etc.)
- The Industrial PC then resubscribes and waits for another notification

Problem 2: Polling for test results

- How do the operators know when the fuel leak test has failed?
  - Traditional approach: UHF radio calls (aka, walkie-talkie) are used to verbally call out problems
  - Alternative approach: Computer generated internal emails
- Radio calls are fast but unaccountable, unreliable, and unrecorded.

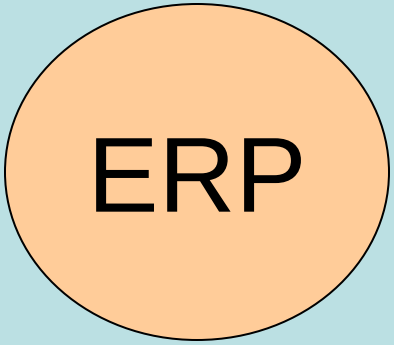
Alternative: Triggers

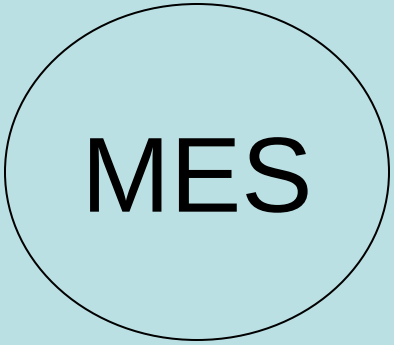
- There is no PC for the test results, so the database needs a different mechanism for notifying people
- Whenever a failure occurs, the database trigger sends an internal email
- iPods or other mobile devices are used to tell the plant supervisors to deal with the failures

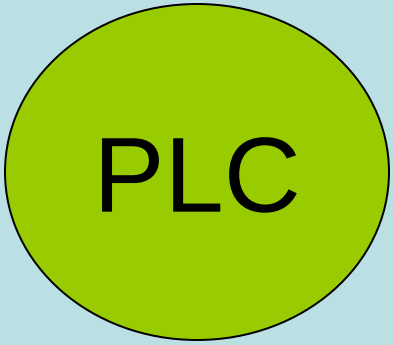
Problem 3: Polling for problematic incoming orders

- How do the operators know when there is a problem with an order that has just arrived?
  - Traditional approach: Manual inspection (or just wait until it causes a problem!)
  - Alternative approach: Use emails and message queuing to SAP/ERP system
- Message queue is already part of billing system

More Examples of Notifications







Enterprise Resource Planning

- New parts have arrived
- A part is changing revision levels (AB -> AC)
- A problem has occurred with supplier deliveries

Manufacturing Execution System

- A new build has an impossible combination of parts
- A repair was performed but not double-checked
- Computer hardware has failed (deadman switch)
- A vehicle failed a critical quality test

Programmable Logic Controllers

- A robot has failed to perform an operation (e.g. assembling tires or stamping a VIN on a vehicle)
- An operator has entered a forbidden area of the plant during production
- Motors are failing to run
- Sensors are reading impossible values

SQL Server Specific Advice

- Must schedule downtime to enable broker
- Stop SQL Server Agent (jobs)
- Switch to single_user
- ALTER DATABASE [Database_name]
  SET ENABLE_BROKER WITH NO_WAIT;
- Fix user permissions for Broker
- Go back to multi_user

Conclusions

- Query notifications are better than polling and readily available in SQL Server (or Oracle) databases
  - think differently about client design
  - use triggers with caution
- Manufacturing Execution Systems are large and complex
  - many points of failure
  - lots of legacy hardware and software
  - great opportunities for query notifications

Questions?

Protocol oriented message mgmt.

- Just try once:
  - Sender just sends each message once, doesn’t care if client receives it
  - Client doesn’t acknowledge messages, and it might not even put them in any particular order or remove duplicates or detect corruption
  - Optimized for speed and simplicity, not reliability
  - Analogous to UDP, streaming video, etc.

Message mgmt. 2

- Server with state (at least once delivery):
  - Server assigns numbers to messages and requires an acknowledgement of each message number.
  - Client still doesn't maintain state, but it does send acknowledgements. Delivery is guaranteed if server resends messages.
  - Application must ignore duplicate messages (idempotency).
  - E.g., Advanced Message Queuing Protocol

Message mgmt. 3

- Both server and client have state (exactly once delivery):
  - Both server and client have message numbers, and server must retransmit unacknowledged messages within the current window.
  - Client must perform reordering, remove duplicates, and detect corruption with checksums.
  - Latency can become a problem, but exactly once delivery is guaranteed
  - E.g., TCP

Why polling is bad for MES

- MES computers often run a general purpose OS (e.g., Microsoft Windows)
  - Windows is not a real-time OS, so timers aren't guaranteed to tick
  - Programs become unresponsive to OS and to users
  - Polling makes crashes difficult to recover because error handling inside the while loop becomes messy
- Tragedy of the commons
  - Database becomes overloaded with polling queries
  - Database locks get held for extended periods of time
  - Excessive network traffic causes packet loss and latency

Reliability

- Sources of unreliability
  - Polling: timers can be unreliable in Windows; choice of interval is difficult
  - Triggers: very reliable, but possibly harmful to unrelated queries
  - QNs: very reliable, but be careful about handling network socket errors
- Suggestions
  - Always do simulations and testing before deployment!
  - Always have a backup plan