

PERFORMANCE TUNING WITH WAIT STATISTICS

Microsoft Corporation

Presented by Joe Sack, Dedicated Support Engineer

Quick bio and presentation logistics...

DSE in the Premier Field Engineer team, Microsoft

- DSE (Dedicated Support Engineer) – responsible for product support (in my case, SQL engine) for enterprise customers (who generally purchase support in 400 hour increments up to 1600 for a FTE)
- 3 years with Microsoft, and have had a 12 year LTR with SQL Server

Work locally in Minneapolis (although most of my team is in Texas)

- Majority DSEs work remotely and visit customers throughout year
- Some are local (like me) and get embedded with a customer

Q&A at end of presentation

- Didn't get to ask your question? Email: josephsa@microsoft.com
- I will post this deck to my blog @ <http://blogs.msdn.com/joesack>

Agenda

What are wait statistics? (2005/2008 perspective)

What to collect? How to collect?

Where to look for wait type info, what to ignore, and what to pay attention to...

Top 10 Patterns from the field (from my experience)

* And before we start a quick – why am I presenting this?

“Must have” references...

Tom Davidson (Microsoft) and the Microsoft SQL Customer Advisory Team wrote articles and white papers on this subject for both SQL Server 2000 and 2005:



Microsoft White Paper “SQL Server 2005 **Waits and Queues**”



Microsoft White Paper “Troubleshooting Performance Problems in SQL Server 2005” (2008 version released this week)

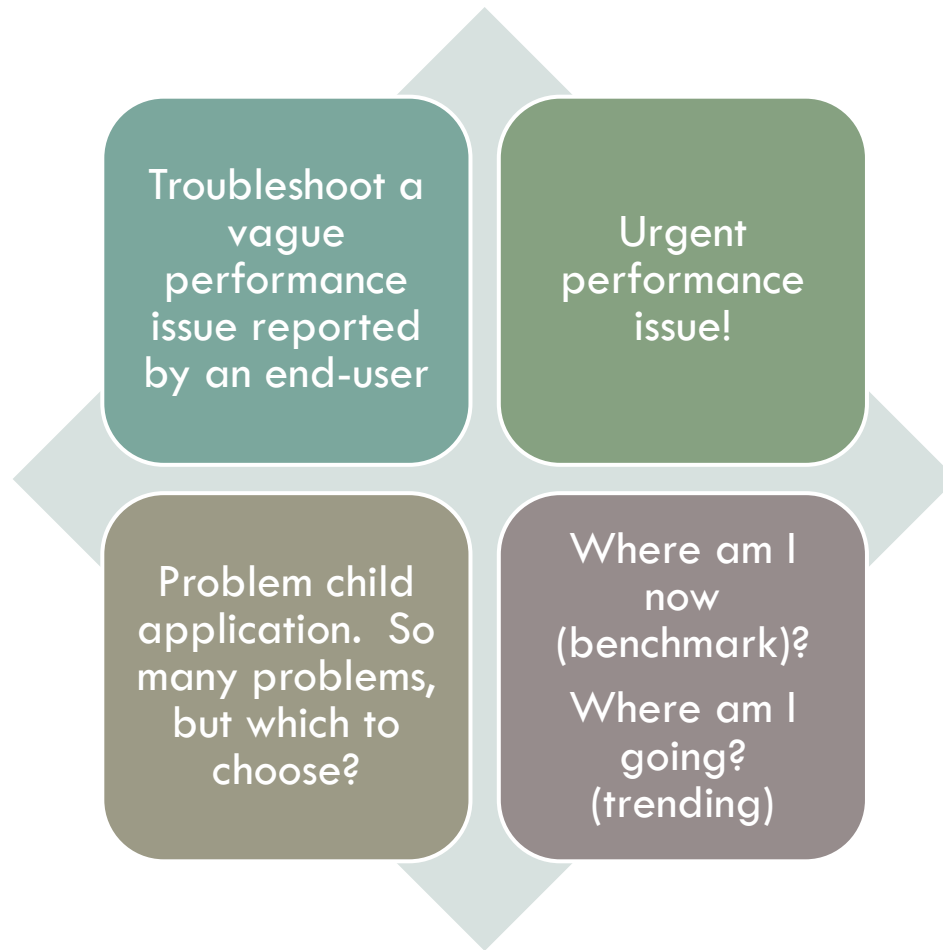


Books Online



<http://support.microsoft.com/>

Usage Scenarios



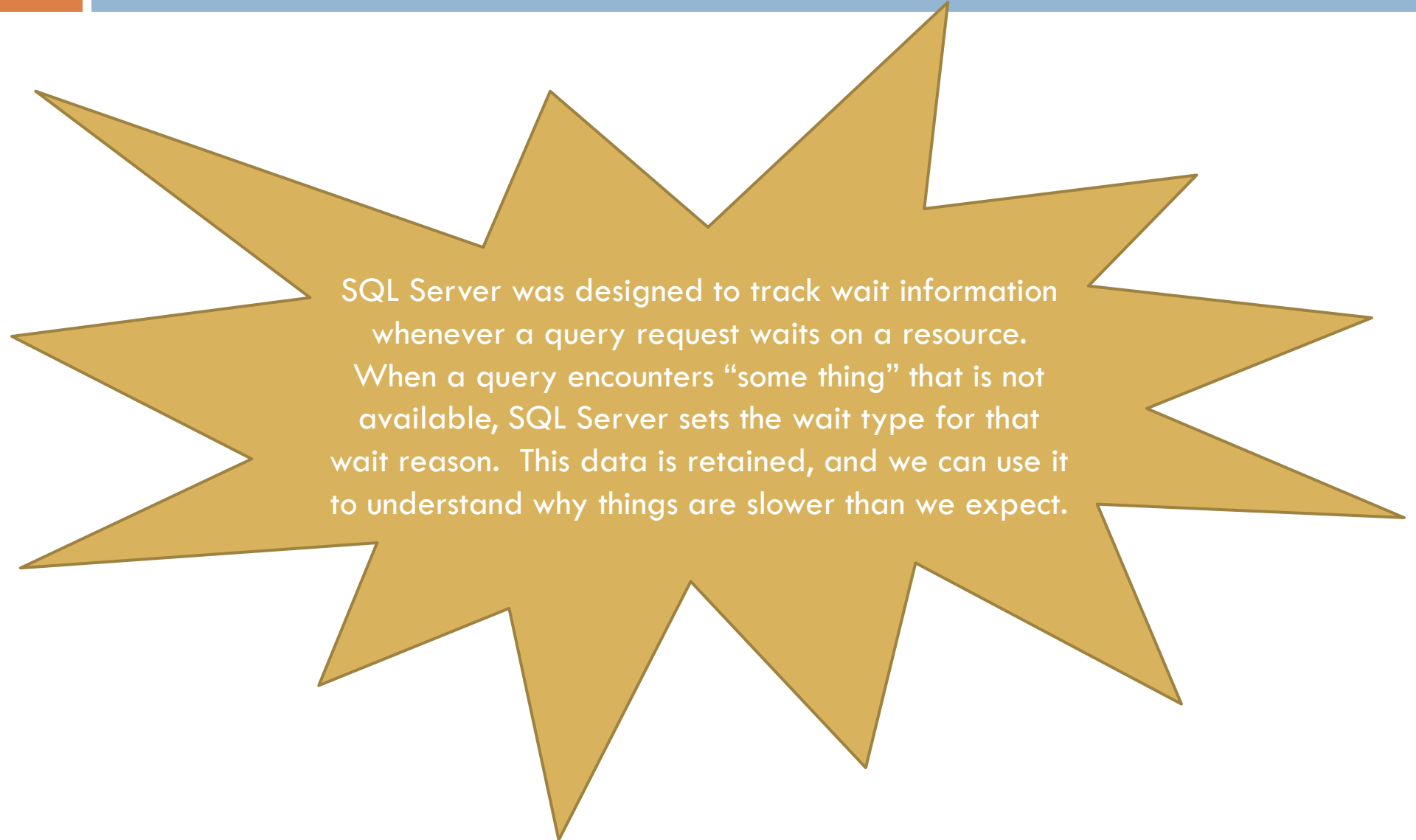
Benefits of looking at Wait Stats...





What are SQL Server Waits?

Wait Statistics in a nutshell

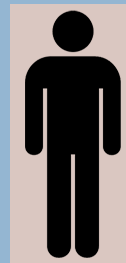
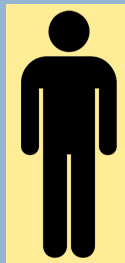
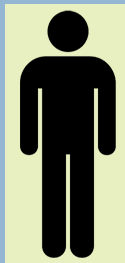


SQL Server was designed to track wait information whenever a query request waits on a resource. When a query encounters “some thing” that is not available, SQL Server sets the wait type for that wait reason. This data is retained, and we can use it to understand why things are slower than we expect.

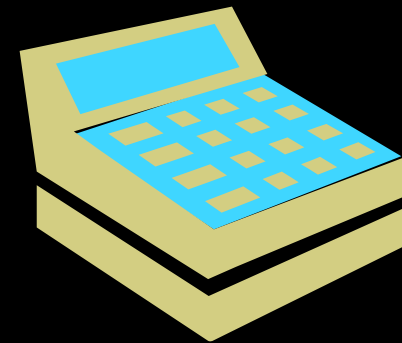
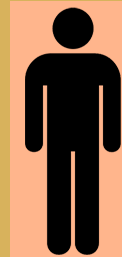
Execution Model – One Scheduler (logical CPU)

One Cashier
at a grocery
store = One
Scheduler

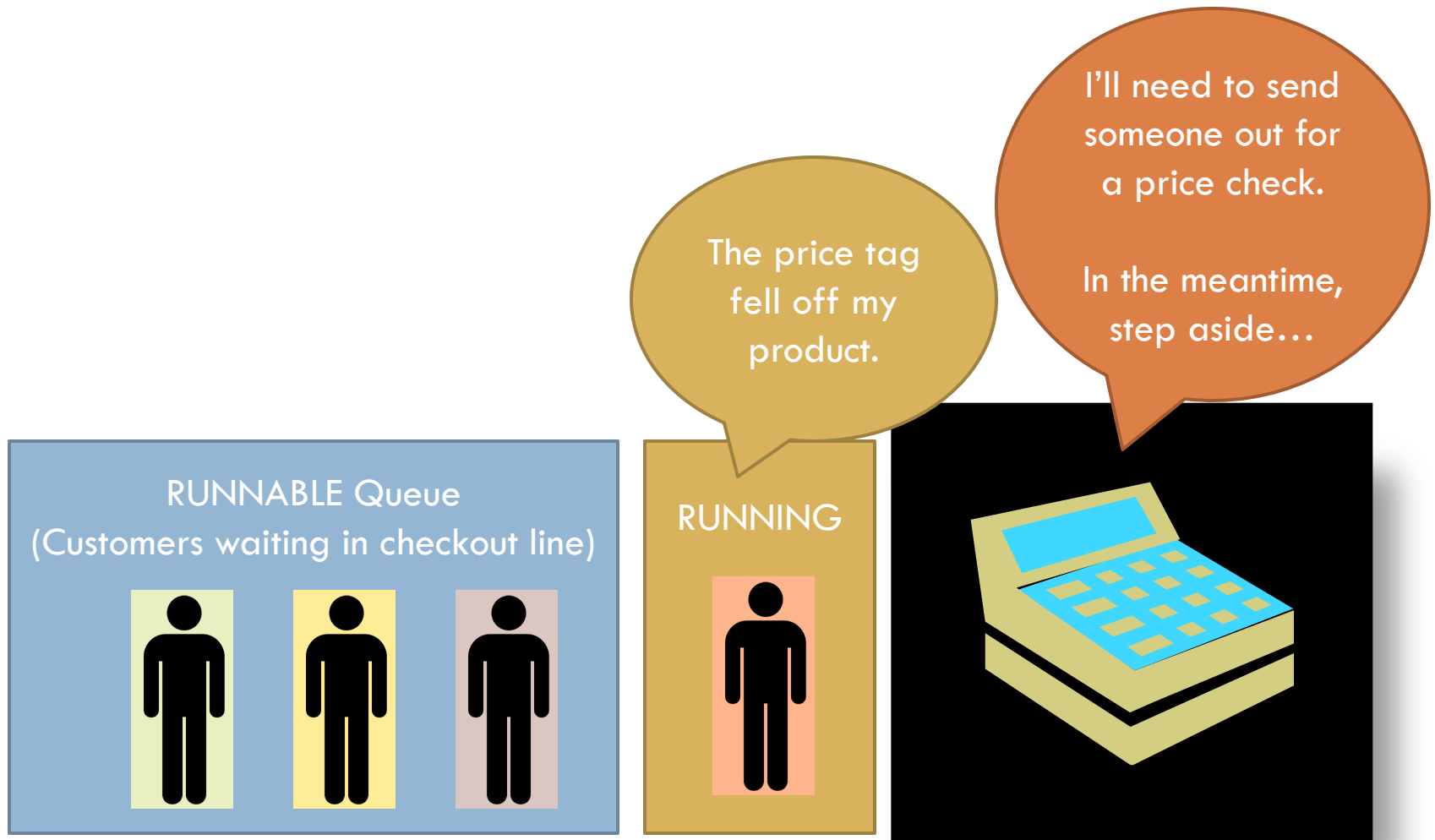
RUNNABLE Queue
(Customers waiting in checkout line)



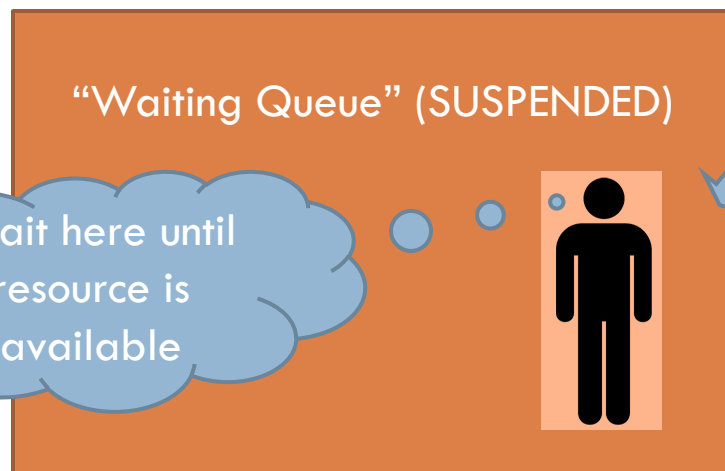
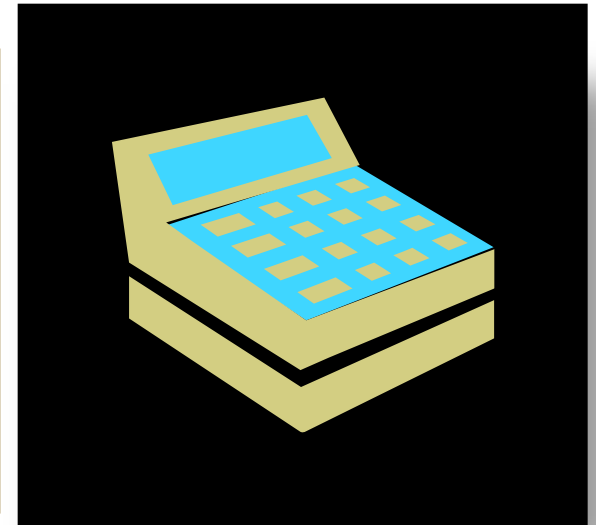
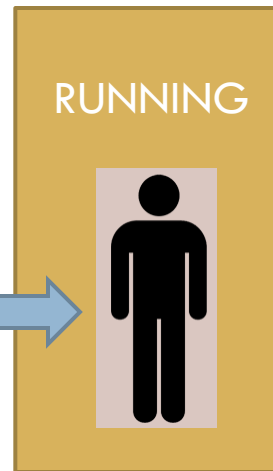
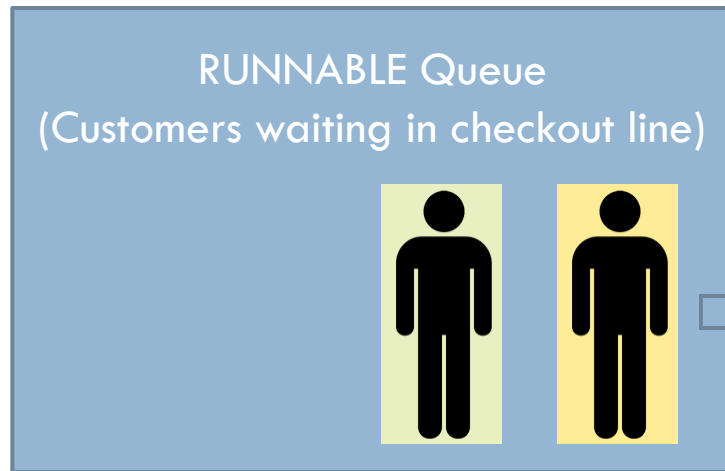
RUNNING



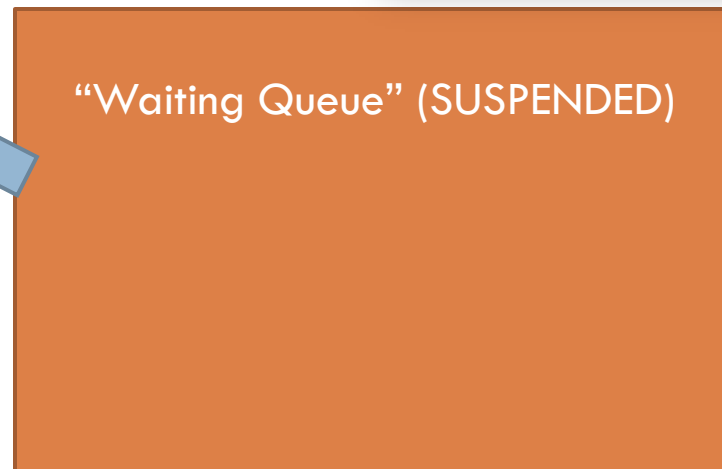
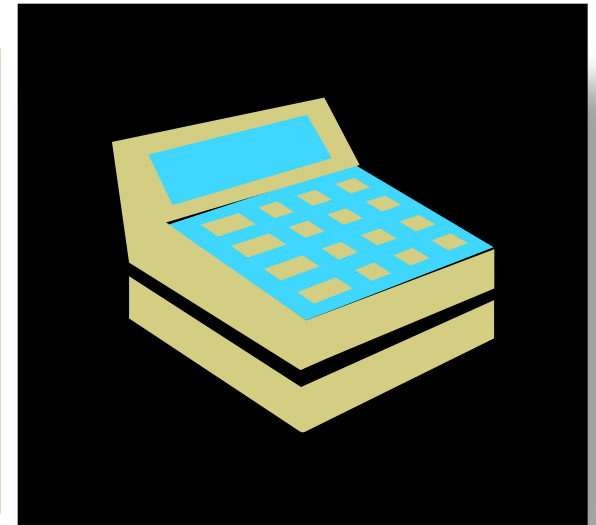
Execution Model – One Scheduler



Execution Model – One Scheduler

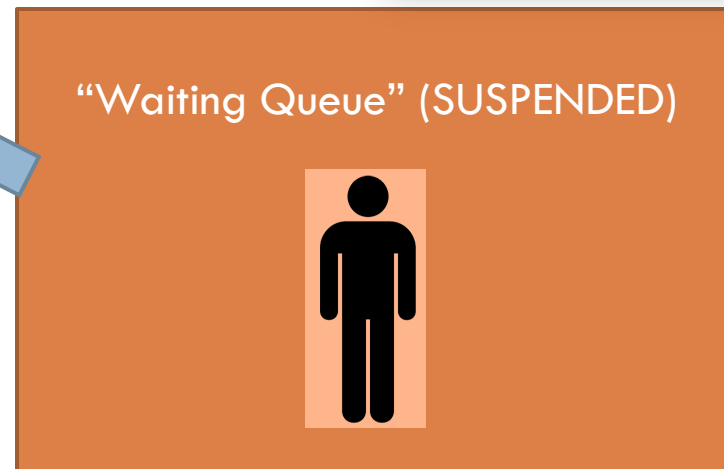
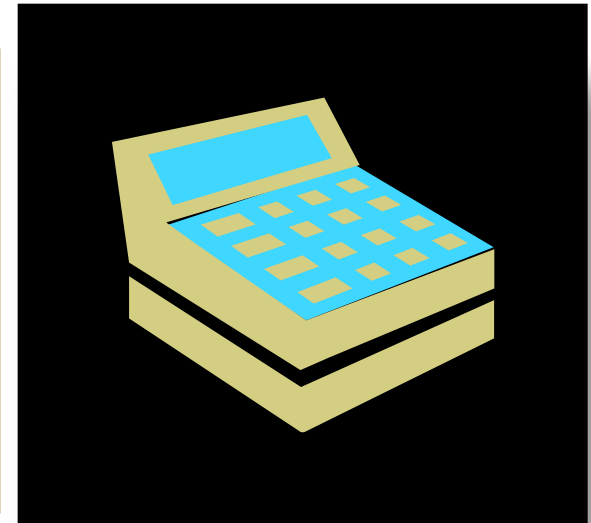
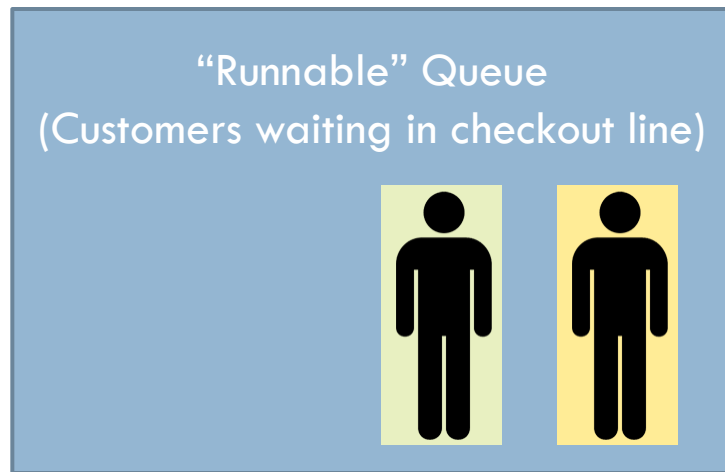


Execution Model – One Scheduler



When I’m done waiting, I go to the end of the runnable line (signal wait time)

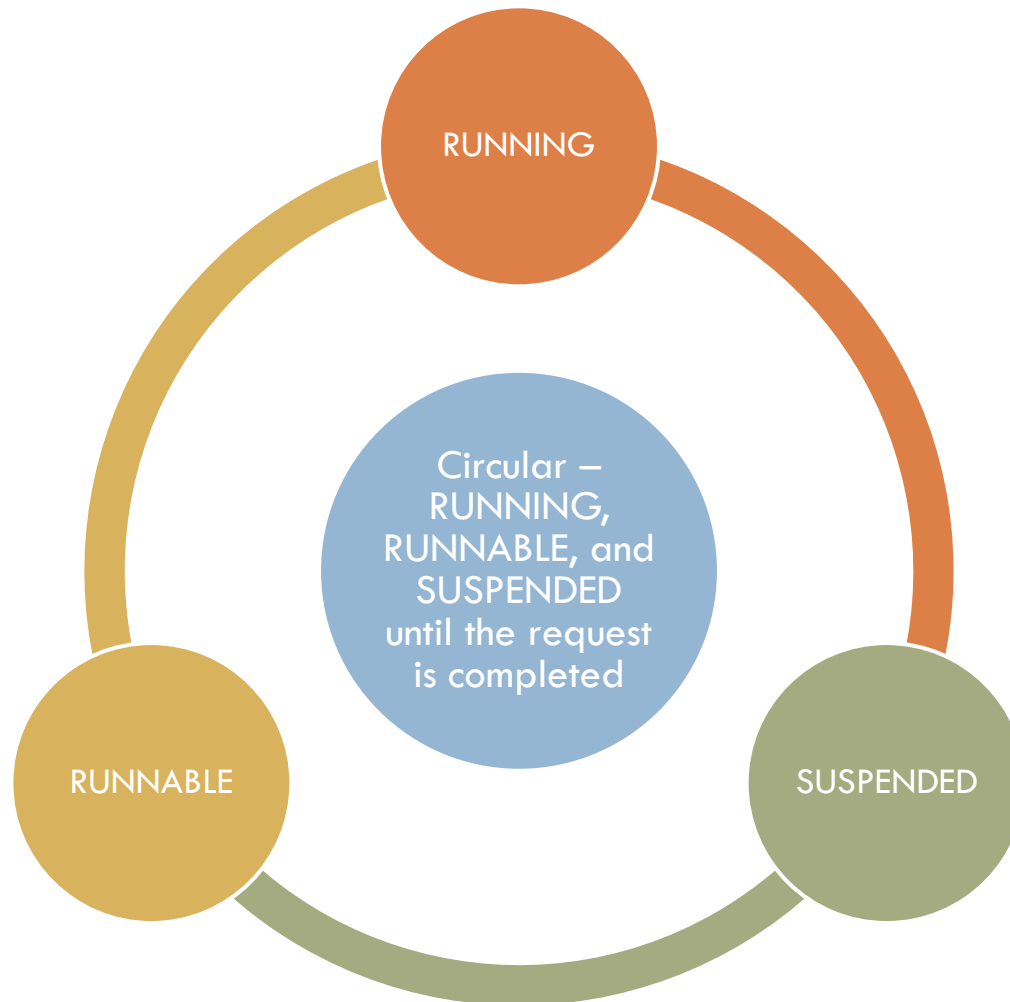
Execution Model – One Scheduler



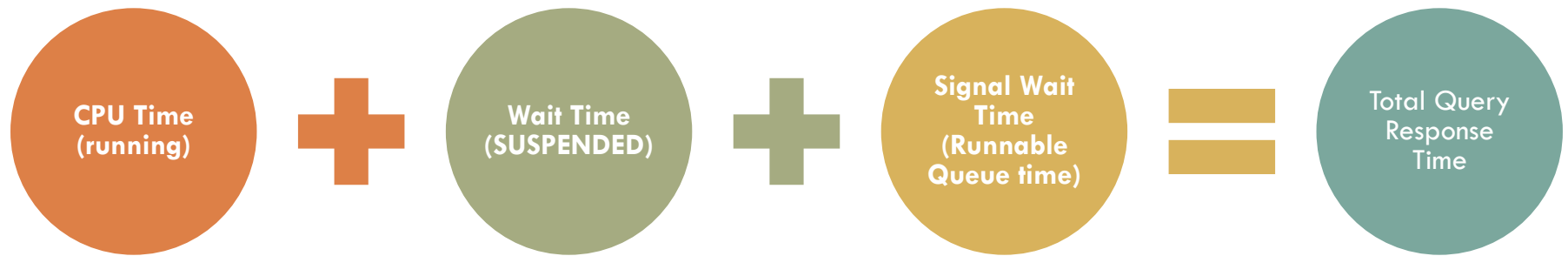
Got sixteen CPUs? It gets a little more active...



Simplified Query Lifecycle...



Total Query Response Time – breaking it down...



Why Waits *and* Queues?

Queues refers to PERFMON counters and other resource usage information

Provides a full picture for application performance

- This presentation focuses on Waits
 - Why? Because most DBAs use Perfmon, but not everyone looks at wait stats (although the word seems to be getting out)...



How to collect? What to collect?

Viewing Wait Statistics at the SQL Instance level

SQL Server 2000

- `DBCC SQLPERF(waitstats)`

SQL Server 2005\2008

- `sys.dm_os_wait_stats`

2005\2008

Data accumulated since
last restart of SQL
Server or last clearing...

- `DBCC SQLPERF('sys.dm_os_wait_stats',CLEAR)`

sys.dm_os_wait_stats exposed data

This is your primary DMV for collecting wait stats information. There are various collection procs you can use from the Microsoft White paper as well...

- ❑ **wait_type** – the name of the wait type
- ❑ **waiting_tasks_count** – number of waits on this wait type
- ❑ **wait_time_ms** – total wait time for this wait type in milliseconds (includes signal_wait_time).
- ❑ **max_wait_time_ms** – maximum wait time on this wait type for a worker
- ❑ **signal_wait_time** – difference between time the waiting thread was signaled and when it started running (time in runnable queue!)

Reporting does not have to be overly complicated...

```
SELECT *  
FROM sys.dm_os_wait_stats  
ORDER BY wait_time_ms DESC
```

Point in time
snapshot...

	wait_type	waiting_tasks_count	wait_time_ms	max_wait_time_ms	signal_wait_time_ms
1	FT_IFTS_SCHEDULER_IDLE_WAIT	4	237001	60000	0
2	XE_TIMER_EVENT	4	120002	30000	120002
3	REQUEST_FOR_DEADLOCK_SEARCH	22	110006	5000	110006
4	SQLTRACE_BUFFER_FLUSH	27	108006	4000	0
5	LAZYWRITER_SLEEP	113	107872	1001	1
6	SLEEP_TASK	175	55342	1067	48
7	BROKER_TO_FLUSH	53	53252	1025	0
8	ASYNC_NETWORK_IO	26	187	179	34

```
DBCC SQLPERF('sys.dm_os_wait_stats', clear)
```

Then clear the stats
and check again
later...

(Tip – don't let a missing business process get in the way of looking at your wait stats)

Keeping it simple - Get a delta

```
-- Capture point-in-time
SELECT wait_type, waiting_tasks_count,
       wait_time_ms, max_wait_time_ms,
       signal_wait_time_ms
INTO #OriginalWaitStatSnapshot
FROM sys.dm_os_wait_stats;

-- Wait for X amount of time
WAITFOR DELAY '00:00:02';

-- Collect again
SELECT wait_type, waiting_tasks_count,
       wait_time_ms, max_wait_time_ms,
       signal_wait_time_ms
INTO #LatestWaitStatSnapshot
FROM sys.dm_os_wait_stats;

-- Compare the results
SELECT l.wait_type,
       (l.wait_time_ms - o.wait_time_ms) accum_wait_ms
FROM #OriginalWaitStatSnapshot o
INNER JOIN #LatestWaitStatSnapshot l ON
       o.wait_type = l.wait_type
WHERE l.wait_time_ms > o.wait_time_ms
ORDER BY l.wait_time_ms DESC
```

SLEEP_TASK	106392
BROKER_TO_FLUSH	103472
LCK_M_IS	61450
ASYNC_NETWORK_IO	1540

Other T-SQL based Methods

From W&Q White Paper

- **Track_waitstats_2005**
 - stored procedure that captures wait statistics from sys.dm_os_wait_stats and provides ranking of descending order based on percentage.
 - Allows number of samplings every X number of minutes

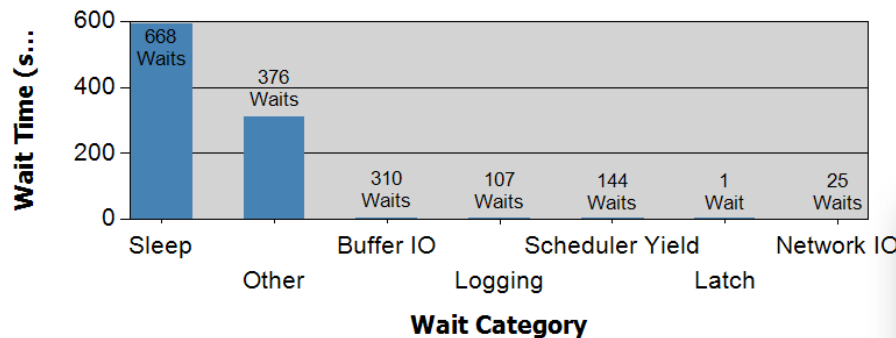
Home grown – you can filter out wait types that are ignorable – for example, filtering out background operations (more on this later)

Performance Dashboard (2005)

Historical Waits

Report Time: 3/20/2009 9:00:12 AM

Cumulative Wait Time By Wait Category



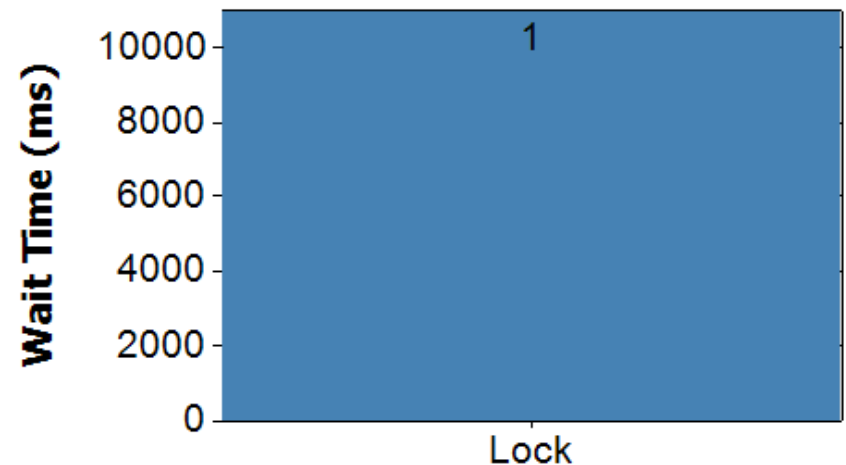
Wait Category	Number of Waits	Wait Time (sec)	% Wait Time
<input checked="" type="checkbox"/> Sleep	668	594.129	65
<input checked="" type="checkbox"/> Other	376	312.233	34
<input checked="" type="checkbox"/> Buffer IO	310	5.724	0
Wait Type	Number of Waits	Wait Time (sec)	% Wait time
PAGEIOLATCH_SH	254	4.57	79.84%
PAGEIOLATCH_UP	20	0.624	10.90%
PAGEIOLATCH_EX	36	0.53	9.26%
<input checked="" type="checkbox"/> Logging	107	1.325	0
<input checked="" type="checkbox"/> Scheduler Yield	144	0.078	0
<input checked="" type="checkbox"/> Latch	1	0.015	0
<input checked="" type="checkbox"/> Network IO	25	0	0

Waits in the Sleep category are typically delays incurred by background tasks waiting for more work. A high wait time in this category is usually not indicative of any performance problem.



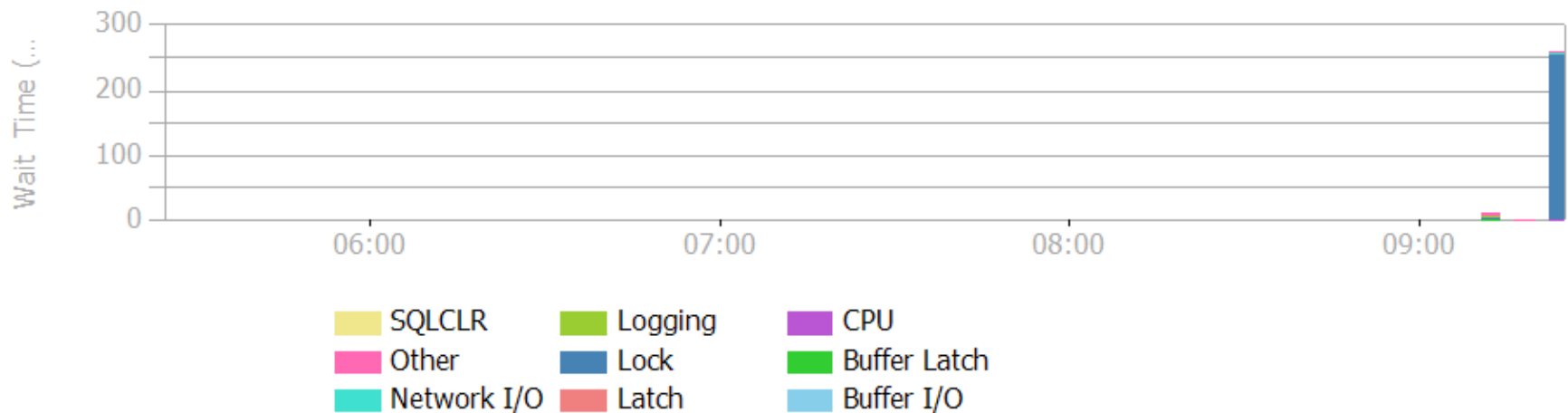
System performance may be degraded because of excessive waits happening on the server. Click on a Wait Category data point in the chart below to investigate further.

Current Waiting Requests



Management Data Warehouse and Data Collector (2008)

SQL Server Waits



Wait Category		Completed Waits	Wait Time (ms/sec)	% of Total Wait Time
<input checked="" type="checkbox"/> Lock		6	84.500	93.37%
	LCK_M_U	2	126.653	93.30%
	LCK_M_X	3	0.065	0.07%
	LCK_M_S	0	0.000	0.00%
	LCK_M_SCH_M	1	0.000	0.00%
	LCK_M_SCH_S	0	0.000	0.00%
<input checked="" type="checkbox"/> CPU		1,252	2.255	2.49%

Request waits

My favorite –
“HELP – what’s
happening!”
DMV

Use `sys.dm_os_waiting_tasks` to show the waiter list at the current moment

- Highly recommended DMV to query during an issue – can reveal interesting patterns (example – app team thinks this is blocking – but we find that WRITELOG is to blame)
- TIP - `sys.dm_exec_requests` contains all requests (running, runnable, waiting)

sys.dm_os_waiting_tasks example

```
SELECT w.session_id,  
       w.wait_duration_ms,  
       w.wait_type,  
       w.blocking_session_id,  
       w.resource_description,  
       s.program_name,  
       t.text,  
       t.dbid,  
       s.cpu_time,  
       s.memory_usage  
FROM sys.dm_os_waiting_tasks w  
INNER JOIN sys.dm_exec_sessions s ON  
           w.session_id = s.session_id  
INNER JOIN sys.dm_exec_requests r ON  
           s.session_id = r.session_id  
OUTER APPLY sys.dm_exec_sql_text (r.sql_handle) t  
WHERE s.is_user_process = 1
```

Results		Messages				
session_id	wait_duration_ms	wait_type	blocking_session_id	resource_description	program_name	text
1	53	302550	LCK_M_IS	52	objectlock lockPartition=0 objid=213...	Microsoft SQL Server Manag... SELECT * FROM dbo.t1000

Waiting Tasks – why look?

May reveal
patterns
beyond what
you can see
in the SQL
instance-
level results

- Example – You see blocking/lock based wait types, but you traverse up the chain and see lead blocker is waiting on the disk subsystem
- Example - See pattern of CXPacket across 8 threads for short bursts

Extended Events – XEvent (2008)

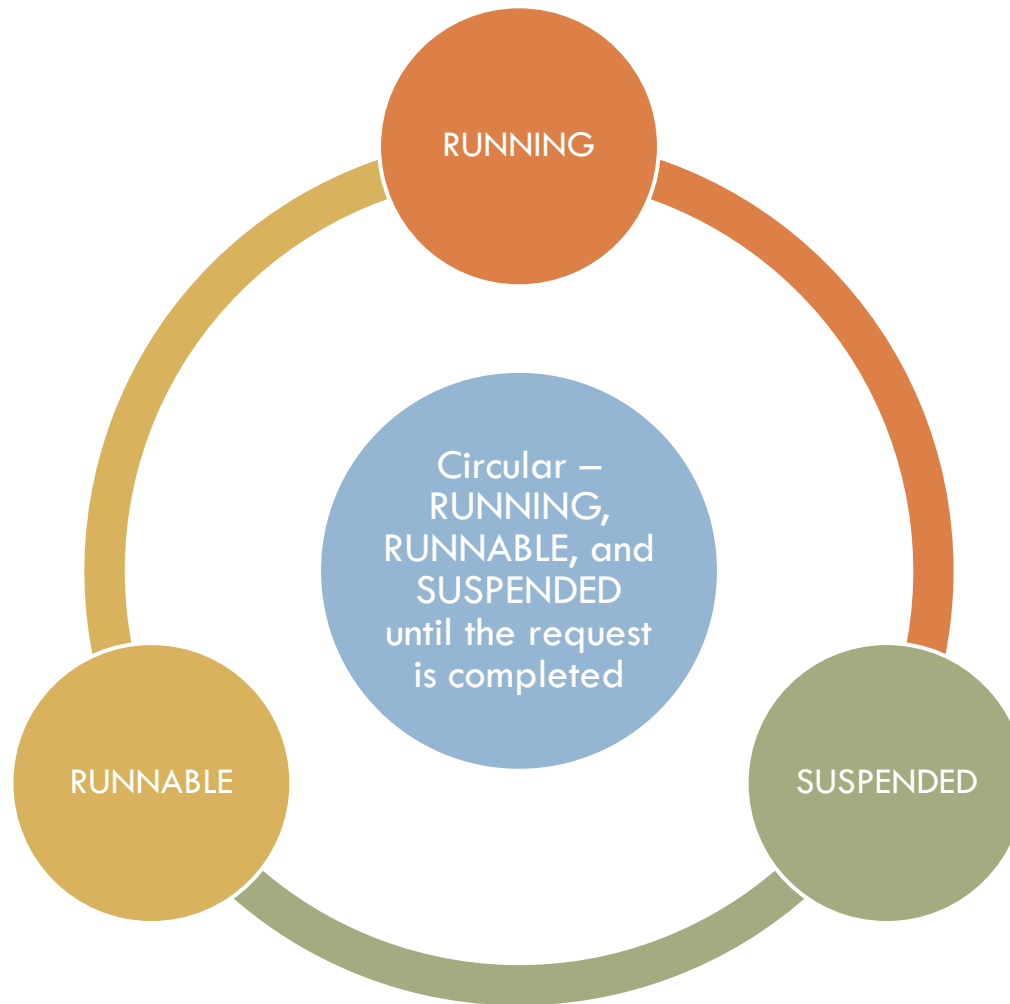
Event-handling system added to SQL Server 2008 for light-weight, low overhead tracing

Help Customer Support by reducing tracing overhead, improving debugging capabilities – all without relying on stack dump or attached debugger

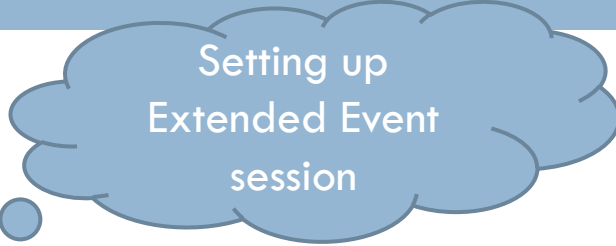
Many use cases – but I talk about it here because it enables the ability to track accumulated wait stats across the lifetime of a request..

* Demonstrated on page 71 of the new “Troubleshooting Performance Problems in SQL Server 2008” White Paper

Flashback...



* Excerpt from Troubleshooting Performance Problems in SQL Server 2008



Setting up
Extended Event
session

```
create event session session_waits on server
add event sqlserver.wait_info
    (action (sqlserver.sql_text, sqlserver.plan_handle, sqlserver.tsq_stack)
where sqlserver.session_id=53 and duration>0)
add target package0.asynchronous_file_target
    (set filename=n'c:\temp\wait_stats.xel', metadatafile=n'c:\temp\wait_stats.xem');

alter event session session_waits on server state = start;
go
```

* See Chapter 2 in Kalen Delaney's new book "Microsoft SQL Server 2008 Internals" for an excellent walk-through of Extended Events.

* Excerpt from Troubleshooting Performance Problems in SQL Server 2008

```
SELECT Name  
FROM HumanResources.Department
```

SPID 53

Query captured
event data

```
SELECT wait_type, SUM(duration) total_duration, SUM(signal_duration) 'total_signal_duration'  
FROM  
    (select  
        CONVERT(xml, event_data).value('(/event/data/text)[1]', 'nvarchar(50)') as 'wait_type',  
        CONVERT(xml, event_data).value('(/event/data/value)[3]', 'int') as 'duration',  
        CONVERT(xml, event_data).value('(/event/data/value)[6]', 'int') as 'signal_duration'  
    from sys.fn_xe_file_target_read_file  
        (N'C:\temp\wait_stats*.xel', N'C:\temp\wait_stats*.xem', null, null)) xyz  
GROUP BY wait_type
```

	wait_type	total_duration	total_signal_duration
1	LCK_M_IS	31282	1
2	NETWORK_IO	174	0

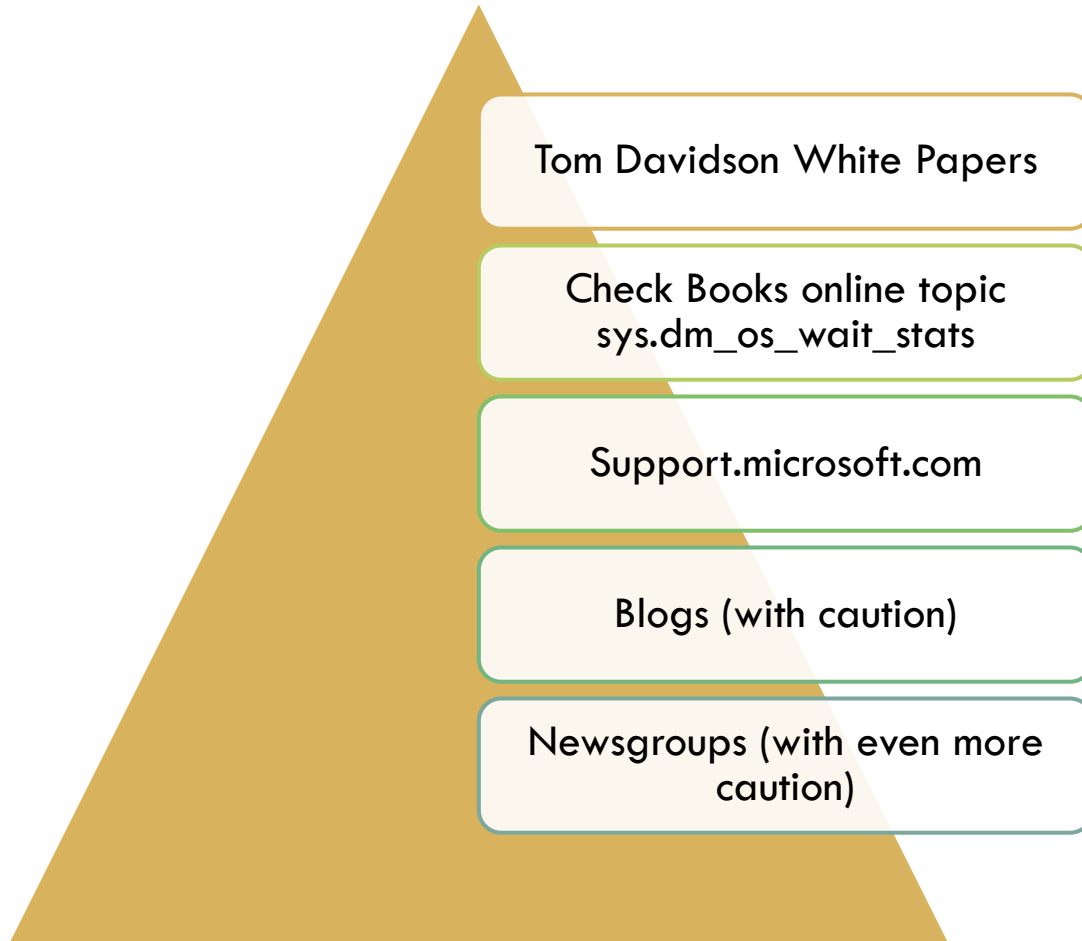
SPID 53 wait type
capture for query

How to interpret?

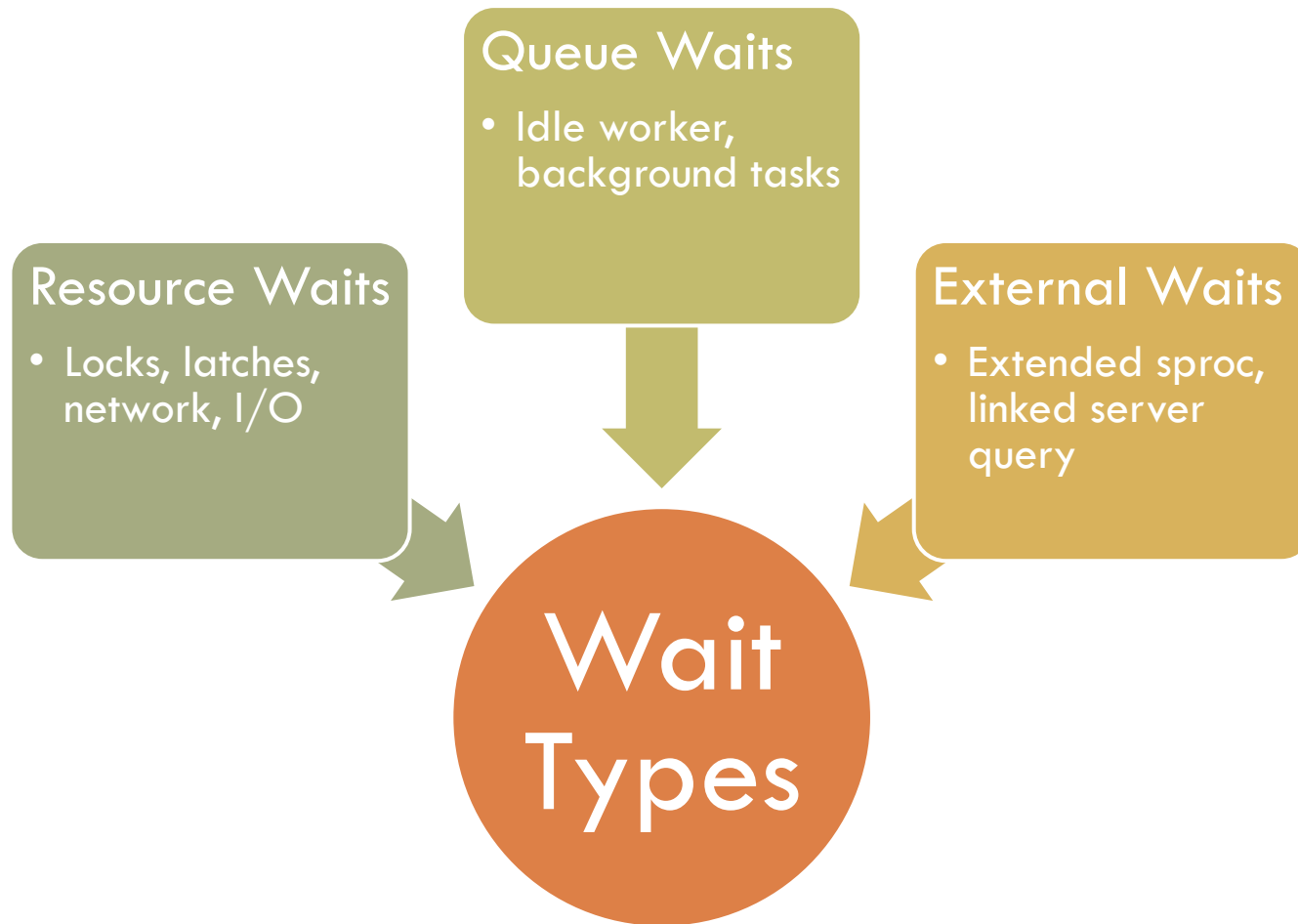
“Never memorize what you can look up in books.”

Albert Einstein

Interpret Wait Type Definitions



Wait Types



Interpreting Wait Types

Not all wait types indicate issues or are actionable
(queue waits – background tasks)

Concentrate on **resource** or **external** wait times or
wait counts, which can point to bottlenecks or hot spots

- I/O (slow response times)
- CPU
 - Some trickiness here – so I'll discuss this more in a bit
- Memory
- Locking (contention by queries)
- Network

In my line of
work – 99.999%
of my attention is
on resource
waits...

Nuances of CPU bottlenecks

Waits don't always reveal CPU bottlenecks directly

- CXPACKET? Yes – could be a parallelism issue
- SOS_SCHEDULER_YIELD? Yes – workers having to frequently yield to other workers may suggest CPU pressure

When a wait type is not waiting or runnable, it is running (and that's good)

- High signal wait time? May need faster or more CPUs

SQL Server 2008 Added Preemptive Wait Types

Non-preemptive (cooperative) scheduling

- SQL Server manages CPU scheduling for most activity (instead of the OS)
 - SQL decides when a thread should wait, or get switched out
 - SQL developers also sprinkle code with voluntary yields – to avoid starvation of other threads

Preemptive scheduling

- Preemption is act of an OS temporarily interrupting an executing task, involuntarily
 - Higher priority tasks can preempt lower priority tasks
- When this happens - can be expensive
- Preemptive mode used in SQL Server for external code calls, CLR with an UNSAFE assemblies, APIs that could block for extended periods (example – extended stored procedures)

Interpreting Preemptive Wait Types (2005 versus 2008)

For cooperative (non-preemptive) tasks, if task is waiting, status of request is set to **SUSPENDED** and wait type is updated based on resource being waited for

SS 2005, a preemptive request (for example, calling an extended stored procedure) would still have a status of **RUNNING** and no associated wait type (not much visibility)

SS 2008, a preemptive request has status of **RUNNING**, and also has a **PREEMPTIVE_XYZ** wait type associated with it

- Extra info helps troubleshoot issues like scheduler hangs (due to external code calls)

* Thank you to Bob Ward, Principal Escalation Engineer for this explanation.

Interpreting Preemptive Wait Types

BOL has some interpretations

Some are undocumented...

Some may be named intuitively

```
SELECT wait_type, waiting_tasks_count  
FROM sys.dm_os_wait_stats  
WHERE wait_type LIKE 'PREEMPTIVE%'  
ORDER BY waiting_tasks_count DESC
```

Results		Messages
	wait_type	waiting_tasks_count
1	PREEMPTIVE_COM_GETDATA	122088
2	PREEMPTIVE_COM_RELEASEROWS	44838
3	PREEMPTIVE_OLEDBOPS	43575
4	PREEMPTIVE_OS_AUTHENTICATIONOPS	24339
5	PREEMPTIVE_OS_DECRYPTMESSAGE	20476
6	PREEMPTIVE_OS_ENCRYPTMESSAGE	15380
7	PREEMPTIVE_OS_LOOKUPACCOUNTSID	8193
8	PREEMPTIVE_OS_AUTHORIZATIONOPS	4089
9	PREEMPTIVE_OS_REVERTTOSELF	4063
10	PREEMPTIVE_OS_DELETESECURITYCONTEXT	4047
11	PREEMPTIVE_OS_DISCONNECTNAMEDPIPE	4047

Preemptive wait type example

Preemptive Wait Type Format:

- `PREEMPTIVE_<category>_<function or class name>`

API function or call can often be found on MSDN

Example:

- `PREEMPTIVE_OS_LOOKUPACCOUNTSID`
 - MSDN explanation of this function “retrieves the name of the account for this SID and the name of the first domain on which this SID is found”

* Thank you to Keith Elmore, Principal Escalation Engineer for this tip!

Interpret based on percentages

- Before going into patterns from the field...
 - ▣ If high percent of wait time is based on ignorable wait types (background threads, for example) – your problem may be elsewhere
 - Furthermore – if signal wait time is not high, then your queries are getting their CPU time and are not waiting on resources
 - You can always further optimize other areas, but you are probably not pushing the system to a bottleneck-state yet



Patterns from the field

Parallelism

Wait Type

- CXPACKET

Interpretation

- Parallel query thread may process a larger number of rows while another thread may process a smaller number of rows

Why worry?

- In OLTP (online transaction processing) system, excessive CXPACKET waits can affect the overall throughput

Next steps?

- For parallelism, **TEST** disabling 'max degree of parallelism' or using MAXDOP hint. Caution about maintenance tasks that use parallelism! (DBCC CHECKDB, index rebuilds)

CPU Pressure

Wait Type

- `SOS_SCHEDULER_YIELD`

Interpretation

- Represents a SQLOS worker (thread) that has voluntarily yielded the CPU to another worker (SQL aims to prevent starvation)

Why worry?

- Is this one of the higher wait types? > 80% of total wait time? May indicate CPU pressure (competing concurrent requests)

Next steps?

- Cross validate perfmon counters. Check `sys.dm_exec_query_stats` by worker time; recompiles/compiles; scale-up or scale-out

Network I/O

Wait Type

- ASYNC_NETWORK_IO

Interpretation

- SQL Server is waiting for the client to fetch the data before it can send any more

Why worry?

- Network issue between the client/application and the SQL Server instance OR your client may be pulling rows inefficiently

Next steps?

- Check network path and components. Network throughput. Client methods for processing all rows. Large result sets with small chunks?

Long term Blocking

Wait Type

- LCK_X (many examples – based on lock type – LCK_M_U (waiting to acquire an Update lock), LCK_M_X (waiting to acquire an Exclusive lock)

Interpretation

- Wait type when task is waiting on a locked resource

Why worry?

- Long term blocking may be unacceptable to your application

Next steps?

- Check transaction duration. Isolation levels (for example – COM using SERIALIZABLE as default). Check perfmon for I/O and memory issues too!

Quick aside - Latches

Latches are light weight synchronization objects

Latches protect the physical integrity of memory structures such as:

- Database pages (Buffer)
- Non-I/O related memory structures

Latches can't be explicitly controlled like locks, but excessive latch waits can cause performance issues are typically tied to contention or resource issues (think SMOKE not FIRE)

Buffer I/O latch

Wait Type

- PAGEIOLATCH_x

Interpretation

- Page IO Latches are used for disk to memory transfers

Why worry?

- When high, could indicate disk subsystem issues or memory issues

Next steps?

- Validate disk and memory perfmon counters. Virtual file stats. SQL logs (for stuck/stall I/O msgs)

Buffer Latch

Wait Type

- PAGELATCH_x

Interpretation

- Latches waiting to access a “Hot” page in memory. Often associated with Tempdb page contention or Clustered Index on Ascending/Descending Key

Why worry?

- May impact performance and represents contention that could likely be resolved. Heavy concurrent inserts into same index range can cause this kind of contention.

Next steps?

- Check if Tempdb allocation page latch contention is the issue (occurs with workloads that create and destroy temp objects) – KB 328551 related to Trace Flag 1118.

Non-buffer Latch

Wait Type

- LATCH_X

Interpretation

- Not I/O buffer related - often seen for internal caches (not the buffer pool pages). Example – LATCH_EX appearing when people use SQL Profiler on remote computer for busy SQL instance.

Why worry?

- Could be an indication of memory pressure

Next steps?

- Validate perfmon memory counters. sys.dm_os_latch_stats DMV to examine details of non-buffer related latches. Likely a downstream side-effect of other performance issues (log related waits or PAGELATCH)

Memory Grants

Wait Type

- RESOURCE_SEMAPHORE

Interpretation

- Query memory request cannot be granted immediately because of other concurrent queries (query gateways). Seen with queries that contain a sort or hash operator.

Why worry?

- Indicates memory pressure. Excessive number of concurrent queries or excessive memory request amount.

Next steps?

- Confirm with perfmon counter counters Memory Grants Pending and Memory Grants Outstanding. Tune queries or determine if memory scale-up needed.

Tran log disk subsystem

Wait Type(s)

- WRITELOG
- LOGBUFFER

Interpretation

- LOGBUFFER - task is waiting for space in the log buffer to store a log record. WRITELOG – task is waiting for a log flush to finish (log flush due to checkpoint or tran commit).

Why worry?

- I/O path or disk subsystem hosting transaction log(s) are encountering performance issues.

Next steps?

- Collocation issues? Validate perfmon and start evaluating disk related counters. Check SQL log for I/O stalls. Check Windows event logs for disk subsystem errors. Check virtual file stats DMV for I/O stall information.

General I/O issues

Wait Type(s)

- ASYNC_IO_COMPLETION and IO_COMPLETION

Interpretation

- Tasks are waiting for I/Os to finish.

Why worry?

- Disk subsystem is likely a bottleneck (not performing to expectations or reaching capacity for given workload)

Next steps?

- Collocation issues? Perfmon (Memory AND I/O). SQL Log for I/O stalls. Event logs. Virtual File stats. Query stats. Indexing. I/O path investigation, SAN, disk alignment, HBA queue depth

Final Takeaways...

When beginning a new performance tuning venture, use wait statistics in your first wave of tools (don't rely too heavily on gut instincts)

White papers give examples of data collection procedures and give good prescriptive guidance. Also consider using 2008's Data Collector functionality or for 2005 the performance dashboard reports

Once you have a direction - drill-down with dynamic management views, logs, SQL Profiler, execution plans, and other tools to troubleshoot

Questions?

“Anyone who knows all the answers most likely misunderstood the questions.”

Anonymous