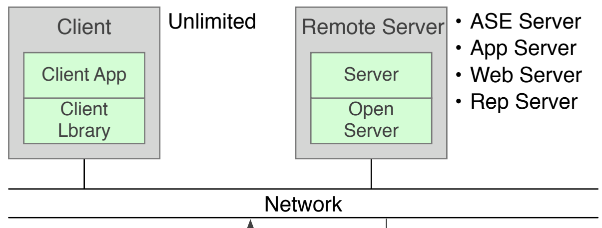
# Foundation

SOFTWARE GEMS

Besides being SQL-compliant, ***Sybase Adaptive Server Enterprise*** is a genuine database server. Unfortunately these days, with cooks and bottle-washers writing postulant "servers", the word *genuine* is demanded. Like Unix, and vastly different to alleged "servers", it:

- consists of a single binary, that runs as pure Multi-Threaded [1] code
- has a Kernel and a Scheduler, which are tightly bound to the o/s and the chipset
- Engines that execute as the *fewest possible* Unix Processes
- designed architected as a database machine, and operates the best on a machine that is thusly suited and configured for the high throughput of a database server




Binary 63MB

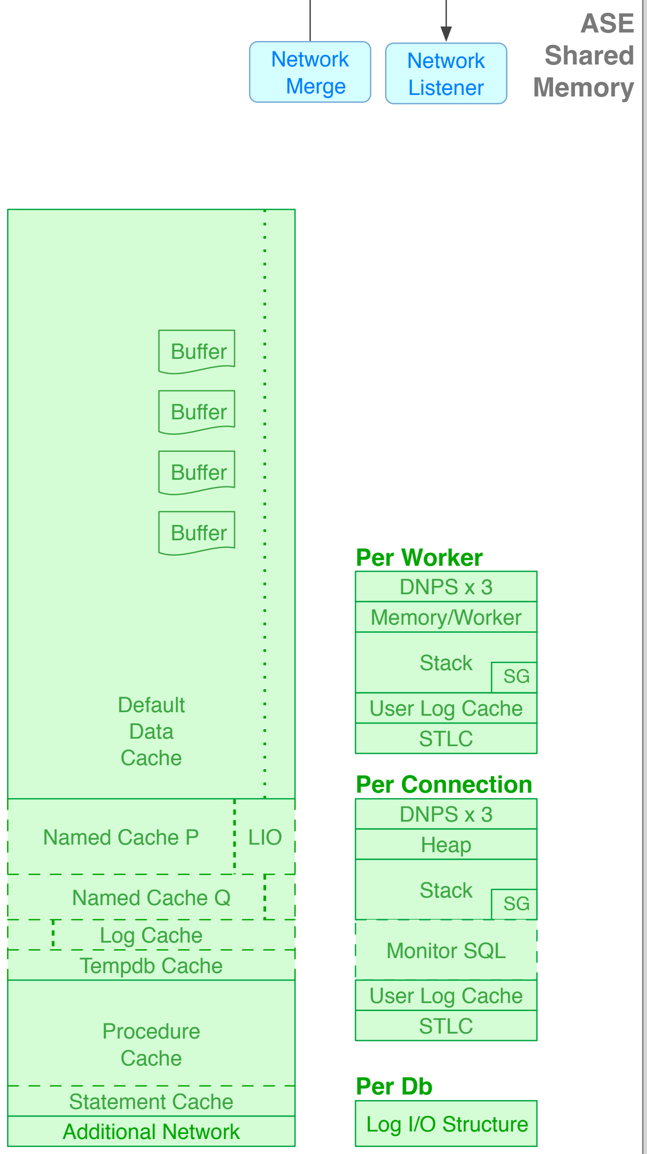
Unix

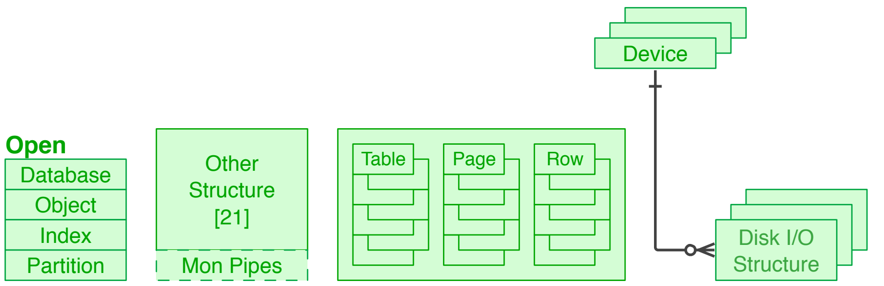
`SERVERNAME.krg`

- All resources are located in a single Shared Memory Segment, allocated, and usually locked
- which is shared by the Engines (Unix Processes)
- the integrity of which is protected by spinlocks (ensure the *fewest* Engines!)
- All resources are highly configurable, and all but the demanded few are dynamic (no reboot required)
- Memory structures that consume considerable space are shown here, the remainder are collected under **Other Structure**. All memory structures are defined later. Dashed lines denote optional structures.
- All larger memory structures are hashed, providing immediate location of objects, and eliminating memory searches. The hash tables are not shown at this level of detail; they are shown in the relevant chapters.
- The number of open objects, etc are collected in **Open**. This is `dbtable`, the highest level memory structure
- Locks shown by Type, but arranged in chains by blocking Task.

### Database Machine

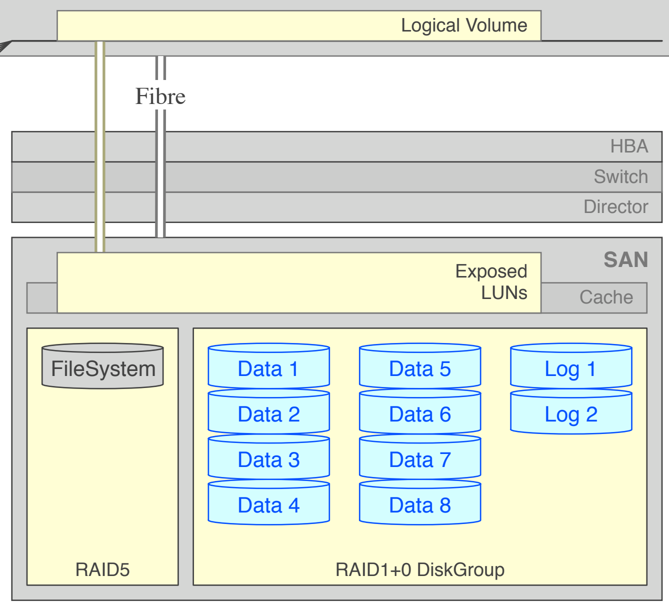
- ***Sybase ASE*** is designed to execute as a Database Machine, a database server that is tightly bound to the host system, the machine
- Performance can be increased dramatically, when
  - the operating system and the machine are configured for database use, and
  - that binding is attended to.
- Unfortunately, that level of performance is not known to new administrators. Because ASE performs much faster than whatever they have migrated from, even on machines that are not configured for database use (default configurations are for generic web servers), greater performance is not sought.




Backplane/IOController/Hub
PCI Card & Extender
Fibre Channel
Local/Remote Device

- Ensure ASE has a clear path to the data
- Ensure the fewest servers compete, at **each layer** in the hardware stack *and the* software stack
- RAID1+0 is recommended over any other
- A Private DiskGroup for each is essential
- Raw Partitions are far superior to /fs files
- *Especially* for tempdb (noting the absurdity of recommendations otherwise)
- Mirroring, to either a local or remote SAN, is provided, as an alternative to SAN-level mirroring




- Physical I/O is the slowest link in the performance chain.
- SAN Configuration is very important for any database server, it is the subject of a 50-page document for customers.

**Green** Memory Component
**Blue** Function

**DNPS** Default Network Packet Size
**STLC** Session Tempdb Log Cache
**SG** Stack Guard

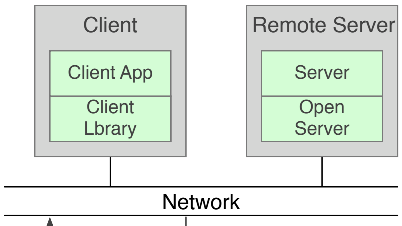
ASE All versions

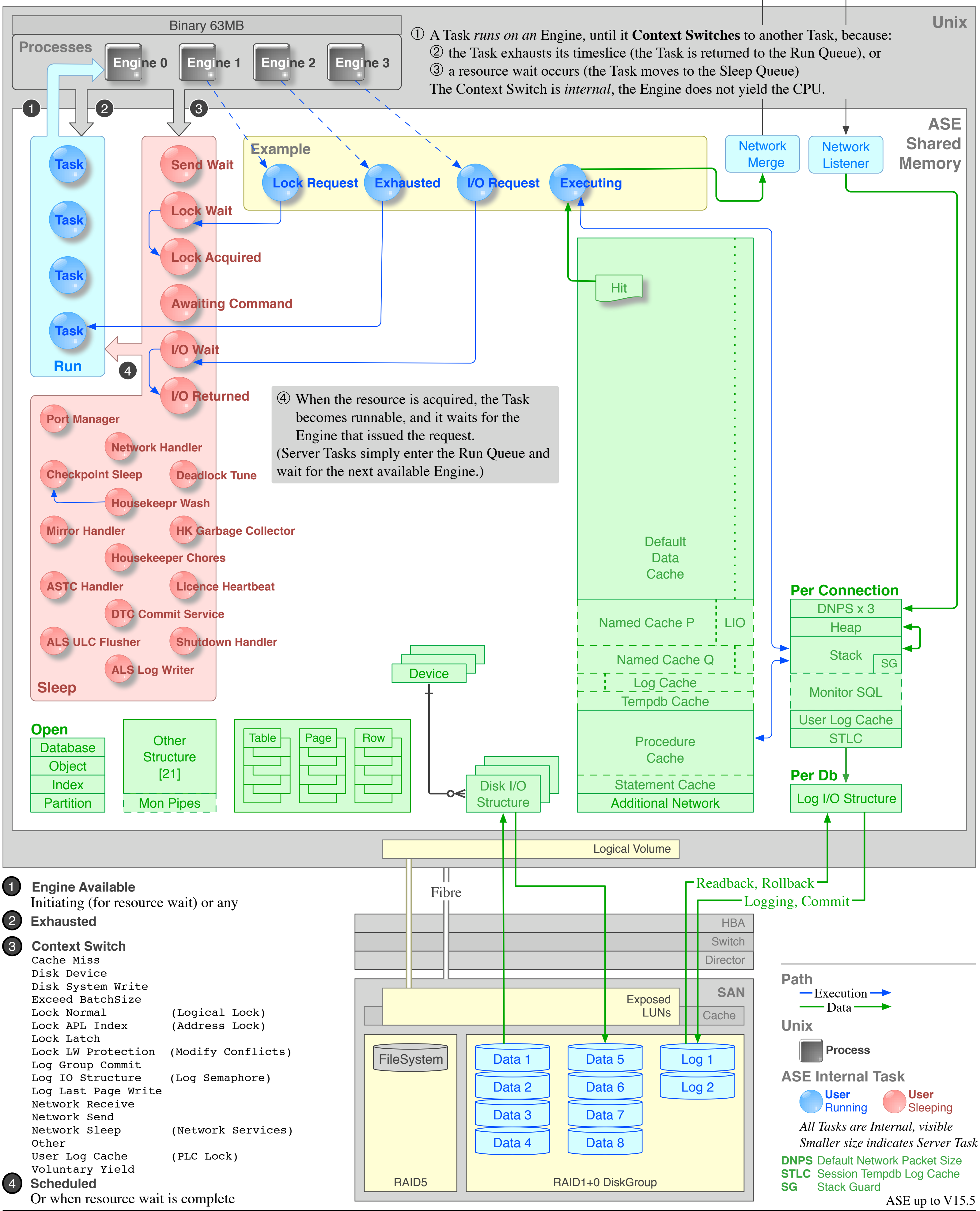
1 The technical term established since 1969, not the private definition used in Oracle documentation. Contrast with Chip Multi-Threading.

# Task Context

- Engines execute as the Unix Processes, which should be the ***fewest possible*** for the load. A Server with four Engines configured is shown
- User Tasks (for each connection and worker) and a few server Tasks (for independent functions), operate as internal processes
- The Example shows states of Tasks, that have been achieved while executing
- 3D articles are live objects. 2D articles are the more fixed context within which they operate

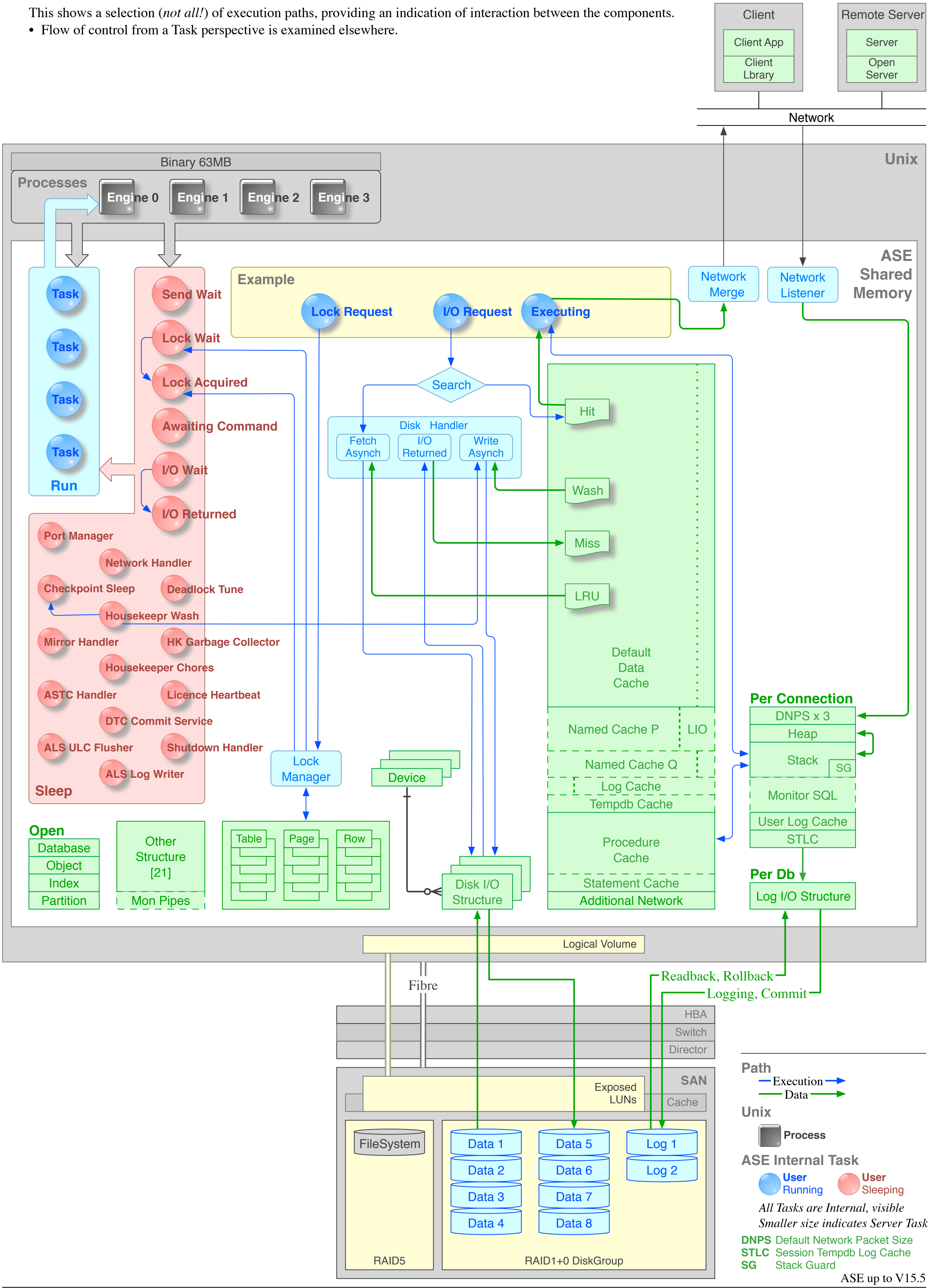


# Sybase ASE Architecture
## Execution

This shows a selection (*not all!*) of execution paths, providing an indication of interaction between the components.

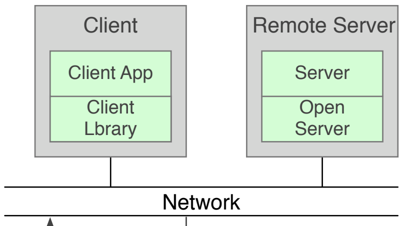
- Flow of control from a Task perspective is examined elsewhere.

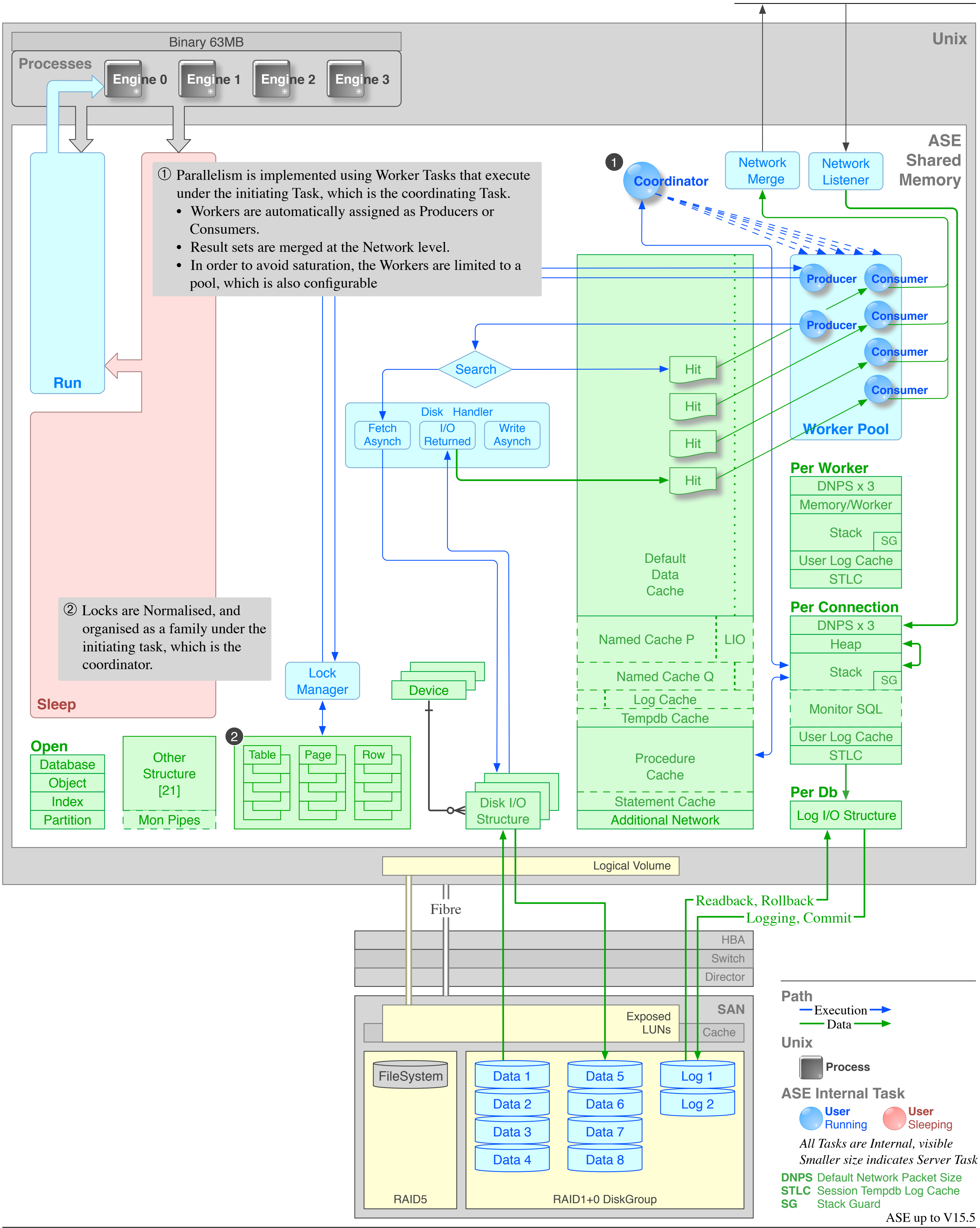


# Parallelism

SOFTWARE GEMS

- ***Sybase ASE*** is massively parallel, not only relating to the machine, but providing it to executing tasks, and that parallelism has been progressed and advanced over decades
- Parallelism can be configured and monitored at **every resource level**



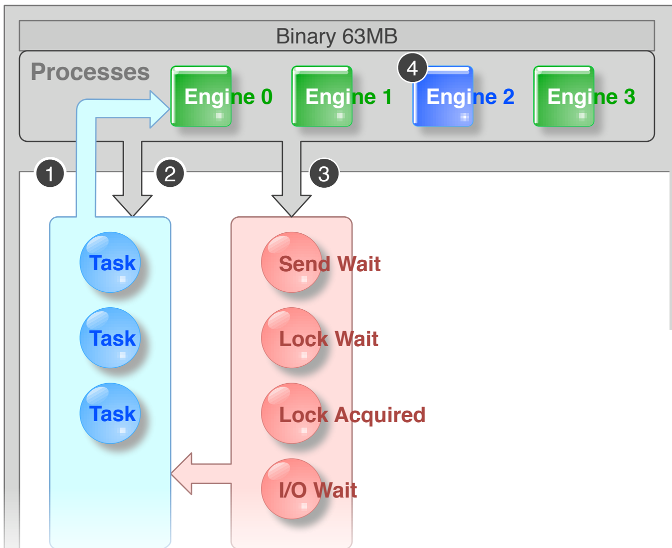



# CPU Yield

SOFTWARE GEMS

This section explains a commonly misunderstood issue, and the compounding negative effects of incorrect machine or ASE configuration.

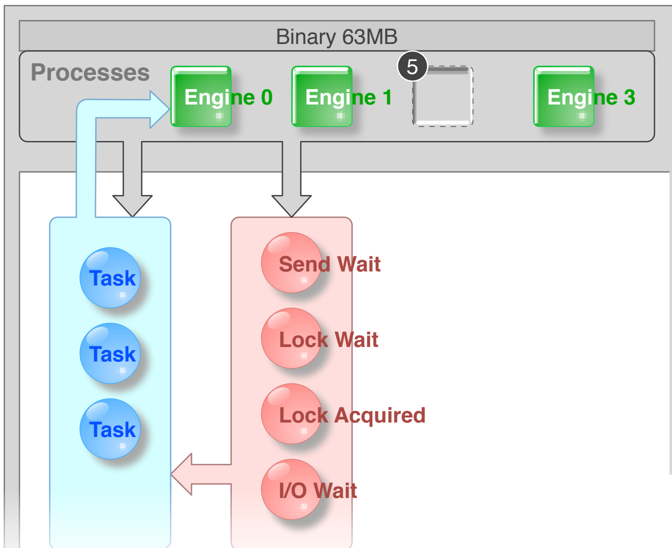
- Note that ASE sees only logical CPUs (whatever Cores or Threads that have been configured on the host system), and it will take full advantage of `max online engines`.




① A Task *runs on an* Engine, until it Context Switches, because:
  ② the Task exhausts its timeslice, or
  ③ a resource wait occurs

The Context Switch is *internal*, the Engine does not yield the CPU. If the Engine finds a task that it can run, the cycle continues. If not, it holds the CPU and waits for one. The method used is:
  ④ it loops `runnable process search count` times.

Eventually
  ⑤ the Engine yields the CPU.

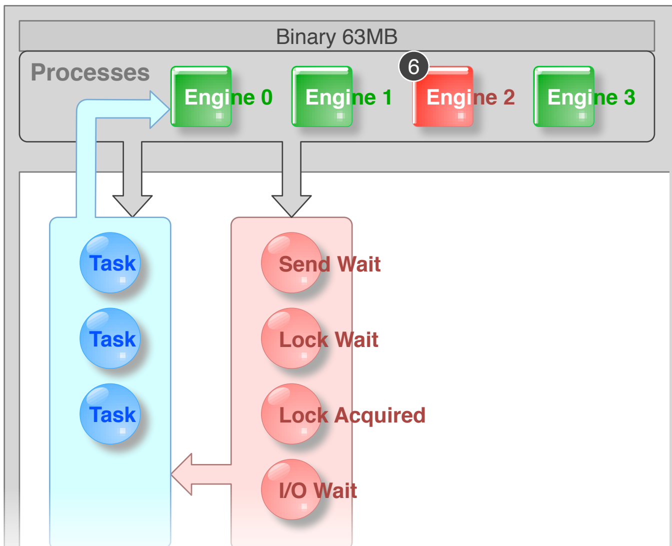



## CPU Yield

***Sybase ASE*** is designed to execute as a Database Server, the only server on the host system, tightly bound to the machine.

**Not Yielding** the CPU is a design principle:

- It avoids o/s level Context Switches, which are the most expensive operation at the Unix level. Once the CPU has been yielded, it has to wait until it is scheduled to execute again.
- *(which is one reason why a product without a* server architecture, *is a circus of clowns, bumping into each other, and why a circus needs a machine ten times more powerful than* Sybase *for the same load).*
- Yielding the CPU is undesirable for a high performance database server
- Note that voluntarily yielding the CPU is quite different to being *forced* off the CPU by Unix, due to other processes (planned or unplanned) running on the system.




## CPU Monopolisation

Failure to understand this, leads to **two common configuration errors**, especially on large machines, each of which, although negative, may be invisible to the untrained eye, but the combination is highly visible to all: high CPU usage *without* a proportionate increase in throughput:

- `runnable process search count` set too high:
  ⑥ Engines are prevented from yielding the CPU when there is no work.
- `max online engines` set too high for the load:
  - over-subscription of CPUs, which means many Engines are very busy doing nothing at all.
  - *Many* Engines are prevented from yielding the CPU.

The two errors in combination results in *many* Engines being busy doing nothing, *and* they are prevented from yielding the CPU.

CPU monopolisation is desirable when the architecture is understood and it is configured appropriately, and it is a disaster when not. *The results of placing people with no tertiary technical qualifications in technical positions are catastrophic.*

## Goal: Fewest Engines, Not Yielding

The architecture is brilliant, it allows tuning of the constituent issues discussed, and it operates uneventfully under the watch of competent Administrators.

- For the machine, configure the fewest Threads per Core; the fewest Cores per CPU, for the load.
- In ASE, configure the fewest Engines for the load.
- Aim for CPU Utilisation to be in the 80-95% range; it is an unyielding Database Server.
- Configure a small `runnable process search count` that is appropriate for your specific machine, o/s and load. Start at 5 or 10, and work upwards.

④ Idle; searching for runnable Task, with limit
⑤ Idle; yielded
⑥ Idle; searching for runnable Task, without limit

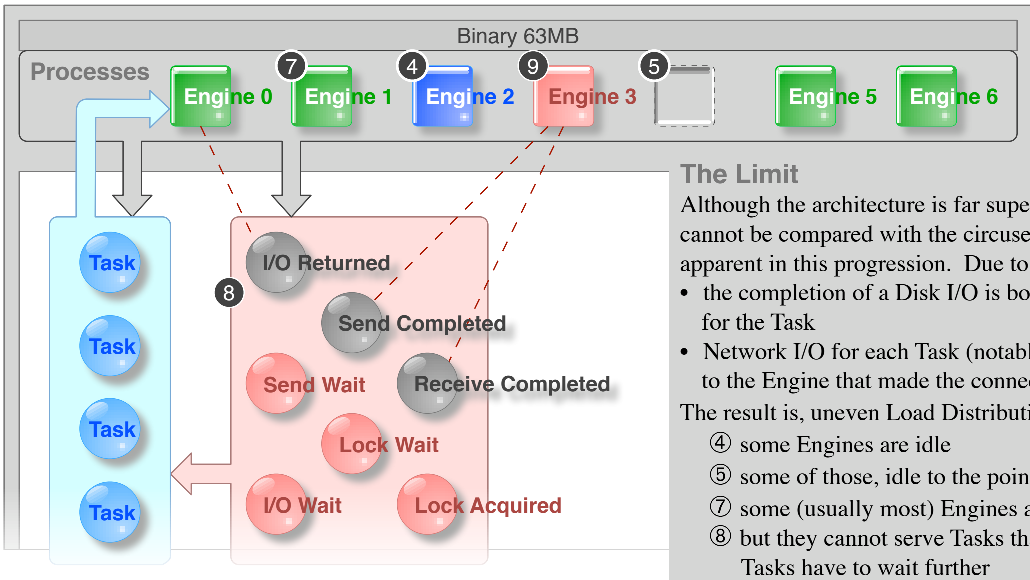
ASE up to V15.5

# Limit

This section explains the limits of the Process Kernel architecture, that motivated the architecture in the next release.

- This should not be confused with the results of configuration errors, described in the previous section.



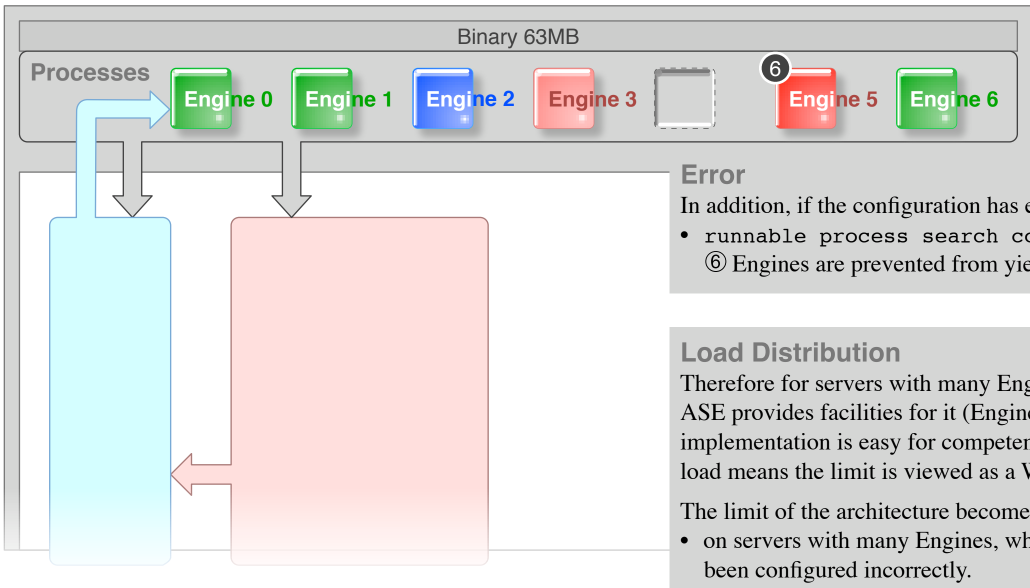

### The Limit

Although the architecture is far superior to alternate architectures (note, it cannot be compared with the circuses), it does have finite limits, which are apparent in this progression. Due to Engines executing as Processes:

- the completion of a Disk I/O is bound to the Engine that issued the request for the Task
- Network I/O for each Task (notably, including RepAgent & CIS) is bound to the Engine that made the connection

The result is, uneven Load Distribution on large servers [2]:

④ some Engines are idle
⑤ some of those, idle to the point of yielding the CPU
⑦ some (usually most) Engines are busy, but not fully utilised [3]
⑧ but they cannot serve Tasks that are bound to other Engines; those Tasks have to wait further
⑨ whilst other Engines are over-utiliaed; the ones that are waited for.




### Error

In addition, if the configuration has errors, such as:

- `runnable process search count` set too high:
  ⑥ Engines are prevented from yielding the CPU when there is no work.

### Load Distribution

Therefore for servers with many Engines **Load Distribution is essential**. ASE provides facilities for it (EngineGroups; ExecutionClasses; etc), and implementation is easy for competent Administrators. Failure to balance the load means the limit is viewed as a Wall rather than an opportunity.

The limit of the architecture becomes visible in two cases only:

- on servers with many Engines, where Load Distribution is absent, or has been configured incorrectly.
- on servers that are over-subscribed (too many Engines for the load), which is a gross error. Here the Wall is encountered prematurely, due to the naïve configuration.

In both cases, the ends of the spectrum (Engines that are idle, or over-utilised) are visible, and the majority in the middle (Engines that are not fully utilised) tends to go unnoticed. It is the Engines in the middle that need more work, that will cancel each end of the spectrum. However, the innocent do not appreciate the relevance of that; they react to the Engines that are over-utilised, and they add more Engines (which are quite useless). Hence the Wall is not only self-created, it becomes insurmountable.

④ Idle; searching for runnable Task, with limit
⑤ Idle; yielded
⑥ Idle; searching for runnable Task, without limit
⑦ Busy; moderate CPU Usage
⑨ Busy; high CPU Usage

2 The architectural limts are experienced only at the high end of throughput, ie. on servers that have a large number of Engines. While smaller systems can operate quite satisfactorily without implementing Load Distribution, such work is demanded for larger systems (anything over 8 Engines). Stated another way, the limit is only a limit due to absence of such work, it is easily overcome..

3. If most Engines are not utilising the CPU to 80 to 95%, the server is over-subscribed, and this will lead to an array of problems (refer to the previous section).

SOFTWARE GEMS

## Sysmon Metric

In order to understand ***Sybase ASE*** and its components, the best avenue by far, is the examination and comprehension of the Metrics reported in `sysmon`. Note that there is, of course, a deep and meaningful, performance-related reason why each Metric is captured, and reported.

Although invaluable, `sysmon` poses problems for some people. A program that processes the reports, such as our ***Sysmon Processor*** overcomes them. Some obstacles posed in `sysmon`, and their manner of address are as follows:

- *Metric names are not consistent across the board, the meaning is sometimes obscured*
- *The organisation of some Metrics in poor. Together, the interpretation of Metrics is hindered*

**Resolution**: Metrics names have been completely Normalised; they are grouped logically and in the relevant hierarchy (notice the indentation), such that it parallels the structure of the server.

- *The reports are difficult to navigate*
- *The Metrics across many reports, which are 40 to 70 pages each, , especially when 24 or 48 are being collected per day, are difficult to correlate*

**Resolution**: The *Sysmon Processor* produces all reports for the day in a grid, and allows various groupings, such as by period, etc. **Example Processed Report**

The structure of the server, and therefore the structure of the processed reports, fall into three categories:

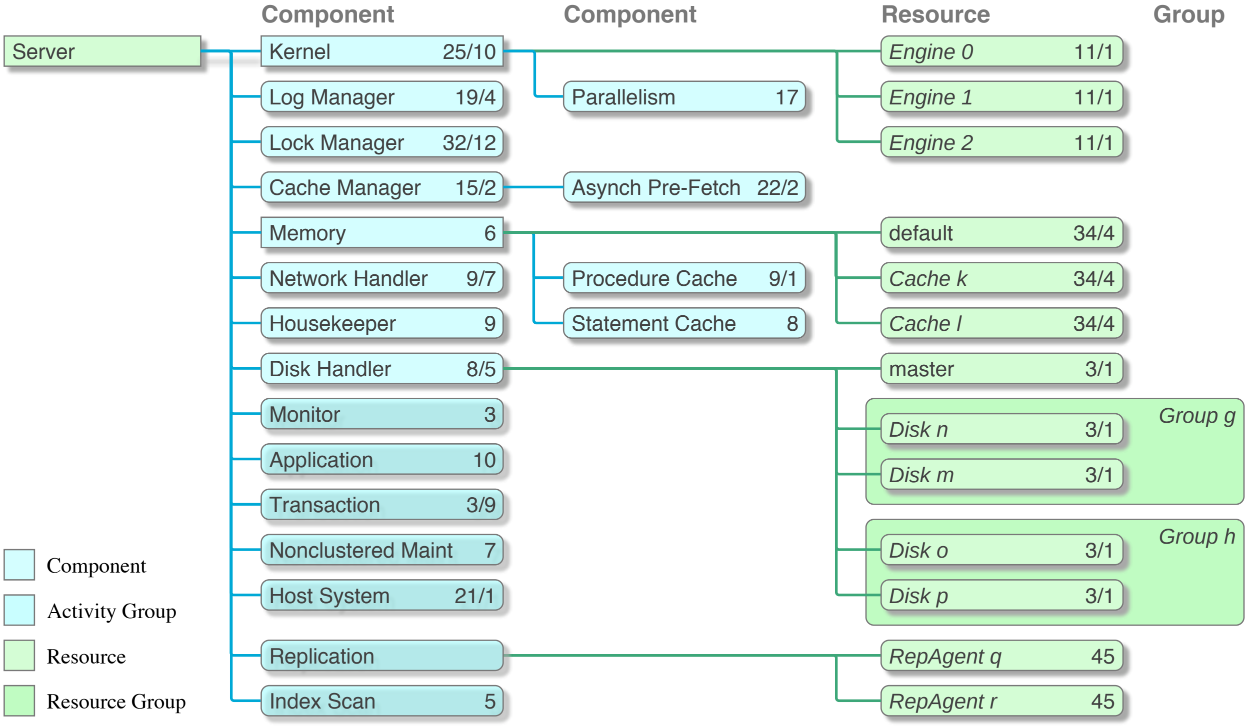
- **Component** Actual components of ASE: these exist in every server, and are available as soon at it is installed. A set of Metrics is collected for each.
- **ResourceType** There are five ResourceTypes: Disk; Cache; Engine, and optionally, ReplicationAgent and Application.
  - **Resource** Of course, everything in the server is a resource, in the normal English sense. Named Resources are specifically those resources that are added by the administrator, after installation; they are specific to each server (the first of each set is added during the installation). A set of Metrics is collected for each Resource. Since there are multiple Resources within each ResourceType, this forms a repeating group of Metrics.
  - **ResourceGroup** Allows Resources to be grouped by usage, type, etc. Essential for large numbers of Resources; and for Load Distribution.
- **Activity** Several Metrics are collected, which are neither Components nor Resources; they are grouped logically, and presented the same as Components.

| Component | ComponentMetric |
|---|---|
| *18 Server Components*<br>`Asynch Pre-Fetch`<br>`Application`<br>`Cache Manager`<br>`Disk Handler`<br>`Host System`<br>`Housekeeper`<br>`Index Scan`<br>`Kernel`<br>`Log Manager`<br>`Lock Manager`<br>`Memory`<br>`Monitor`<br>`Network Handler`<br>`Nonclustered Maint`<br>`Procedure Cache`<br>`Parallelism`<br>`Statement Cache`<br>`Transaction` | *289 Server level Metrics, grouped by Component*<br>`Disk Check`<br>`Disk Check Returned IO`<br>`Disk IO Outstanding`<br>`Disk IO Request`<br>`Disk IO Completed`<br>`...`<br>`Log ULC Record`<br>`Log ULC Grant`<br>`Log ULC Wait`<br>`Log Log IO Structure Grant`<br>`Log Log IO Structure Wait`<br>`Log Log Allocation`<br>`...`<br>`Lock Request Table`<br>`Lock Request Table-Excl`<br>`Lock Request Table-Excl-Int` |

| ResourceType | Resource | ResourceMetric |
|---|---|---|
| `Application`<br>*18 Metrics* | *Server-specific, as configured* | `Appl CPU Busy`<br>`Appl IO Busy`<br>`Appl Idle ...` |
| `Cache`<br>*52 Metrics* | | `Cache Wait`<br>`Cache Search`<br>`Cache Hit ...` |
| `Disk`<br>*4 Metrics* | | `Disk Read APF`<br>`Disk Read`<br>`Disk Write ...` |
| `Engine`<br>*12 Metrics* | | `Engine Busy`<br>`Engine CPU Busy`<br>`Engine IO Busy ...` |
| `RepAgent`<br>*45 Metrics* | | `RepAgent Log Scan`<br>`RepAGent Record Scanned`<br>`RepAgent Log Truncation Wait ...` |

## Structure of Sybase ASE

In order to understand ASE, it is important to understand not only the Components, but their hierarchy within the server:




- *The Host System is of course outside the Sybase server. In order to allow Host System Metrics (*`vmstat, iostat`*) and ASE Metrics to be examined together, and to be charted or graphed together, it is treated as a* `Component`.
- *If it is implemented, Application level metrics are captured, (each Application is treated as a* `Resource`*). Not shown.*
- *While this diagram serves to identify the Structure of ASE, to some degree, and hopefully increases your ability to monitor it and improve its performance, it does not constitute an Architecture or Componentry diagram.*

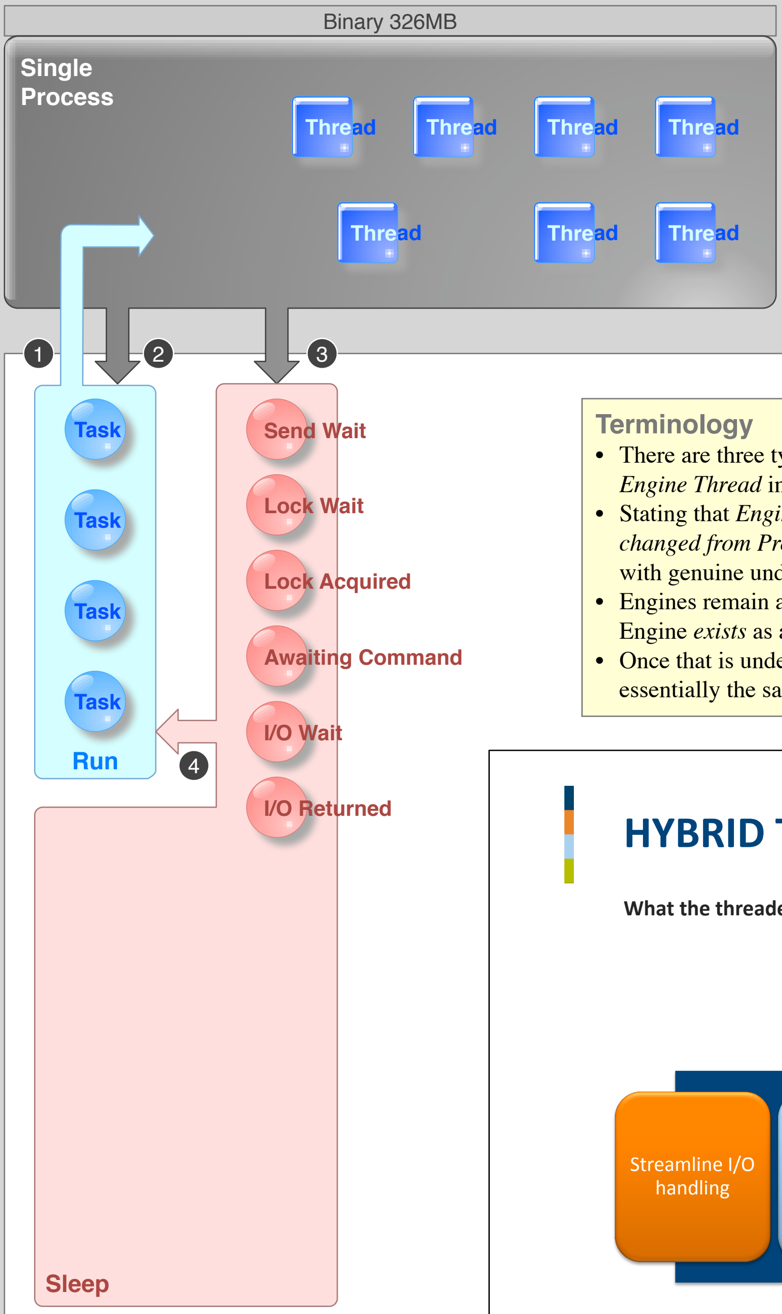
The numbers in the cells identify the number of raw and Computed Metrics captured for the Component or Resource in the current version of our ***Sysmon Processor***.
Additional Metrics are computed at execution time:

- Utilisation, which is provided for all Resource Metrics
- Rate Per Sec for selected Metrics
- Schedule Utilisation

# Threaded Kernel

The K21 Threaded Kernel is the latest progression, in a venerable series of progressions, in both Symmetric Multi-Processing and Chip Multi-Threading:

- Modern operating systems are moving away from multi-process parallelism to multi-threaded processes, fully utilising hardware Threads
- ASE is progressing in the same manner
- It executes as a single Unix Process, using genuine o/s Threads
- The Kernel and its Threads are (no surprise) highly configurable
- Again this should be the ***fewest possible*** for the load.



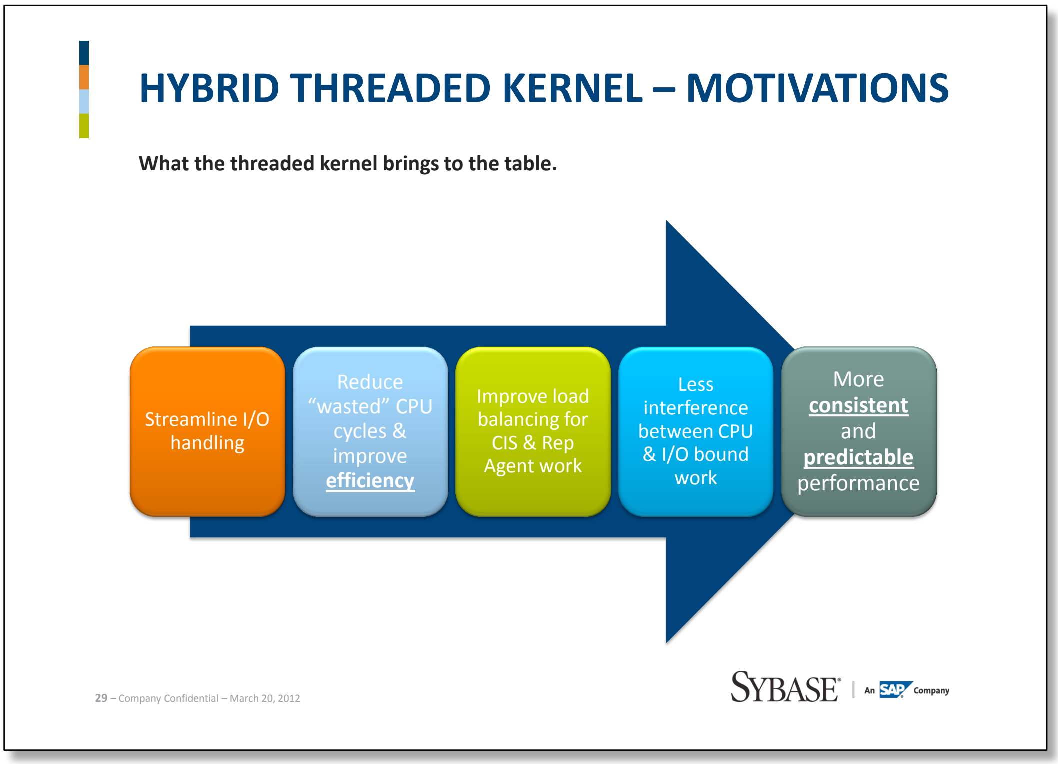

Unix

At the level relevant to Task execution, nothing has changed:
① A Task *runs on an* Engine (now a Thread), until it Context Switches:
② the Task exhausts its timeslice, or
③ a resource wait occurs
The Engine Thread Context Switches *internally* to another Task, without yielding the CPU.
④ When the resource is acquired, the Task becomes runnable.

ASE Shared Memory

**Terminology**

- There are three types of Threads. The documentation uses the terms *Engine* and *Engine Thread* interchangeably, without explanation, which causes confusion
- Stating that *Engines have been replaced with Threads*, or that the *kernel has changed from Process-based to Thread-based*, is too simplistic and interferes with genuine understanding
- Engines remain as tangible articles, with a full set of Metrics. However an Engine *exists* as an Engine Thread.
- Once that is understood, an *Engine* and an *Engine Thread* can be said to be essentially the same thing; the difference is apparent from the context.




① **EngineThread Available**

② **Exhausted**

③ **Context Switch**

```
Cache Miss            Log IO Structure
Disk Device           Log Last Page Write
Disk System Write     Network Receive
Exceed BatchSize      Network Send
Lock Normal           Network Sleep
Lock APL Index        Other
Lock Latch            User Log Cache
Lock LW Protection    Voluntary Yield
Log Group Commit
```

④ **Scheduled**

After resource wait is complete

**Unix**

Process Thread

**ASE Internal Task**

User Running User Sleeping

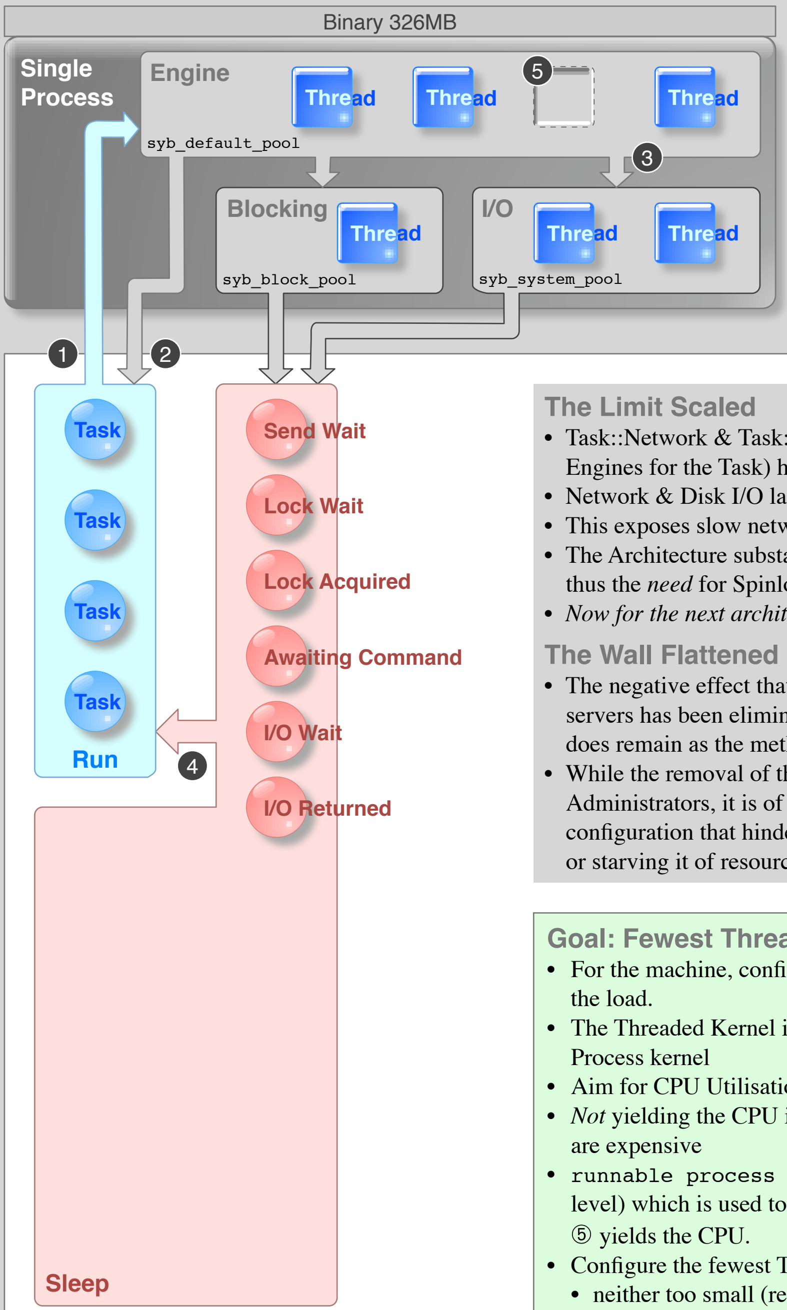
*All Tasks are internal, visible*
*Smaller size indicates Server Task*

ASE V15.7 & subsequent

# Thread Pool

SOFTWARE GEMS

- Three types of Threads, organised into pools
- Additional Threads can be configured in Engine & I/O Pools
- Additional Engine ThreadPools can be configured
- EngineGroups are replaced by Engine ThreadPools




**Unix**

① A Task *runs on an* **Engine Thread**, until it Context Switches, because:
② the Task exhausts its timeslice, or
③ requires I/O, when it is transferred to an **I/O Thread** which issues the request.

The Engine Thread executes *any* runnable Task.

④ When the I/O completes, the Task is picked up by *any* I/O Thread, which handles the completion, etc. The Task is now runnable.

- I/O Threads execute for very short durations.
- **Blocking Threads** are used for server Tasks that Run To Completion.

**ASE Shared Memory**

### The Limit Scaled

- Task::Network & Task::Disk I/O affinity (being bound to the initiating Engines for the Task) has been eliminated
- Network & Disk I/O latency has been eliminated
- This exposes slow networks and SANs *even more acutely than before*!
- The Architecture substantially reduces Shared Memory contention, and thus the *need* for Spinlocks
- *Now for the next architectural limit to be determined !*

### The Wall Flattened

- The negative effect that the absence of Load Distribution has on large servers has been eliminated, such configuration is no longer *essential*. It does remain as the method for Load Distribution at the processor level.
- While the removal of the psychological Wall is Good News for naïve Administrators, it is of course still possible to implement a poor configuration that hinders performance, such as oversubscribing a server, or starving it of resources, or strangling it at the I/O level.

### Goal: Fewest Threads, Not Yielding

- For the machine, configure the fewest Threads per Core; the fewest Cores per CPU, for the load.
- The Threaded Kernel is more advanced, more tightly bound to the machine, than the Process kernel
- Aim for CPU Utilisation to be in the 80-95% range; it is an unyielding Database Server.
- *Not* yielding the CPU is a design principle, it avoids o/s level Context Switches, which are expensive
- `runnable process search count` is replaced with `idle timeout` (ThreadPool level) which is used to hold the CPU while it searches for runnable Tasks. After which it ⑤ yields the CPU.
- Configure the fewest Threads in ASE for the load, and an `idle timeout`. that is:
  - neither too small (results in premature CPU yields)
  - nor too large (results in consuming CPU cycles for nothing, high CPU Usage with no proportionate return).

① **EngineThread Available**

② **Exhausted**

④ **Scheduled**
After resource wait is complete

③ **Context Switch**

```
Cache Miss              Log IO Structure
Disk Device             Log Last Page Write
Disk System Write       Network Receive
Exceed BatchSize        Network Send
Lock Normal             Network Sleep
Lock APL Index          Other
Lock Latch              User Log Cache
Lock LW Protection      Voluntary Yield
Log Group Commit
```

**Unix**
Process Thread

**ASE Internal Task**
User Running User Sleeping

*All Tasks are internal, visible*
*Smaller size indicates Server Task*

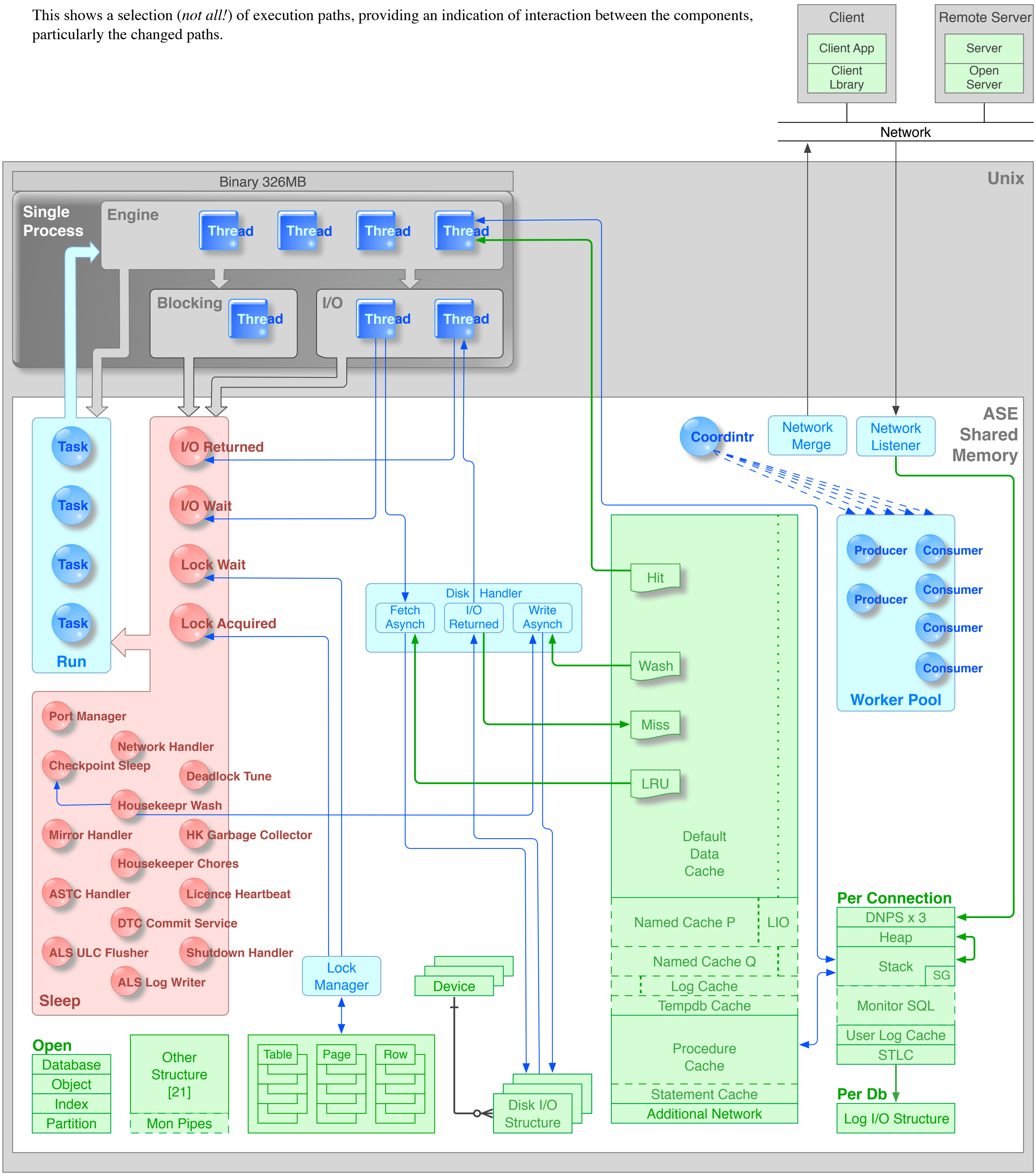
ASE V15.7 & subsequent

1 This is possible in the Threaded Kernel, because all threads belong to a single Process.

## Sybase ASE Architecture

# Threaded • Execution

This shows a selection (*not all!*) of execution paths, providing an indication of interaction between the components, particularly the changed paths.




### Warning

- If your 15.5 configuration was poor, do not migrate the problem to the Threaded Kernel, **the errors will be magnified**. Fix that first.
- If you do not understand the 15.5 Kernel, and cannot configure it properly for your machine and your load, you will not be able to configure the Threaded Kernel. *If you can't ride a highly-strung horse without drama by yourself, it is not reasonable to attempt show-jumping*.
- The consequence of this Kernel being so advanced and powerful, and ASE being so configurable, is that gross configuration errors will cause it to (a) run into race conditions, such as high CPU Usage with little work being completed, or (b) run slower, and with reduced throughput.
- ***Sybase ASE*** is not a circus with hundreds or thousands of clowns, it is a performance by a few star performers, and highly configurable. You must decide what kind of race you are running, and configure the server accordingly. *The configuration required for a sprint vs a marathon vs a 1,500m race, are quite different.*

**Path**
Execution
Data

**Unix**
Process Thread

**ASE Internal Task**
User Running
User Sleeping

*All Tasks are internal, visible*
*Smaller size indicates Server Task*

**DNPS** Default Network Packet Size
**STLC** Session Tempdb Log Cache
**SG** Stack Guard

ASE V15.7 & subsequent