

# Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decision. The development, release, and timing of any features or functionality described for Anaplan's products remains at the sole discretion of Anaplan.

**Introduction**  
**Value**  
**Availability**  
**Deployment**  
**Sparsity & Performance**  
**Scale**  
**Q&A**

# What is Anaplan Polaris?

## What is Polaris?

- Polaris is a new underlying **storage and calculation engine** for Anaplan. It is a **natively sparse engine** – designed as a **general purpose planning and modelling engine** for naturally sparse business problems.
- It will work side by side with the existing ‘Classic’ engine on a workspace by workspace basis. A workspace will either be a ‘Classic Workspace’ or a ‘Polaris Workspace’.
- The main advantages of a natively sparse engine is that it allows much **higher dimensionality and granularity** for sparse business problems.

# Anaplan Polaris

- The same Anaplan modelling interface you currently use...
- With a new underlying storage and calculation engine...
- Allowing massive dimensionality – more dimensions and / or more items in a dimension

The screenshot displays the Anaplan Polaris modelling interface. On the left is a navigation pane with a tree structure. The main area shows a 'Line Item Formula' table for 'Lease Revenue'. The table has columns for 'Formula', 'Parent', 'Is Summary', 'Format', 'Region L2', 'Applies To', and 'Time Seed'. The formulas are organized into sections: 'Income Statement', 'GROSS PROFIT', 'SUBTOTAL EXPENSES', 'ALLOCATIONS', 'OPERATING INCOME', and 'NET INCOME'. Each row contains a detailed formula, a parent item, a summary flag, a format, a region, and a time seed.

The screenshot shows the 'Model Settings' dialog box in Anaplan Polaris. The 'Time' dimension is selected. A red circle highlights the text '2,267,706 trillion cells'. Below this, a 'Model Calendar' section is visible. At the bottom, a summary table shows the dimensions and their respective cell counts.

Dimension	Count
<b>HYPERSPACE</b>	2,267,706,421,290,386,430
<b>DATASPACE</b>	1,074,500,843

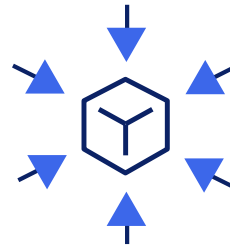
**What value does Anaplan  
Polaris bring?**

## With Anaplan Polaris, businesses can...



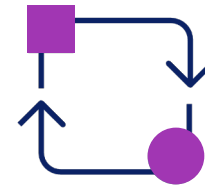
**Broaden and deepen  
the scope of  
addressable business  
problems**

- ✓ Solve business problems by modeling data in its natural state
- ✓ Analyze multiple intersections of highly sparse data for more granular insight
- ✓ Expand data dimensionality of list items to plan and scale with business



**Let business users analyze  
and report at scale,  
unassisted, intuitively and  
easily**

- ✓ Empower business users to slice and dice data for in-depth analysis and reporting
- ✓ Eliminate the need to concatenate or/and flatten data structures, hierarchies, split models (by regions, for example) for an intuitive user experience



**Lower total cost of  
ownership (TCO) in  
a unified platform**

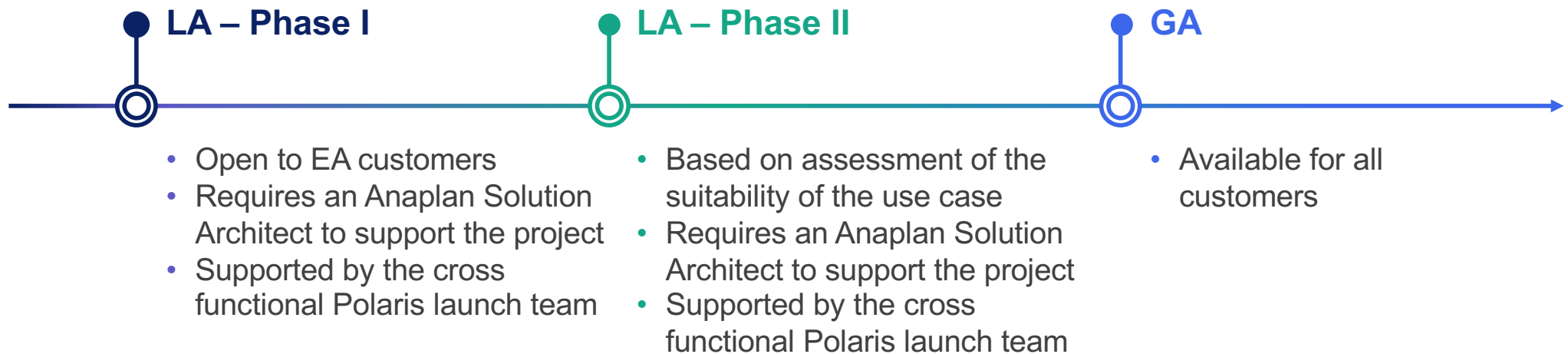
- ✓ CAPEX: Consolidate the software estate by reducing or eliminating multiple planning and analysis tools
- ✓ OPEX: Reduce the extra resources and time needed to manage and maintain multiple tools

**When will Polaris be available and  
what functionality is supported ?**



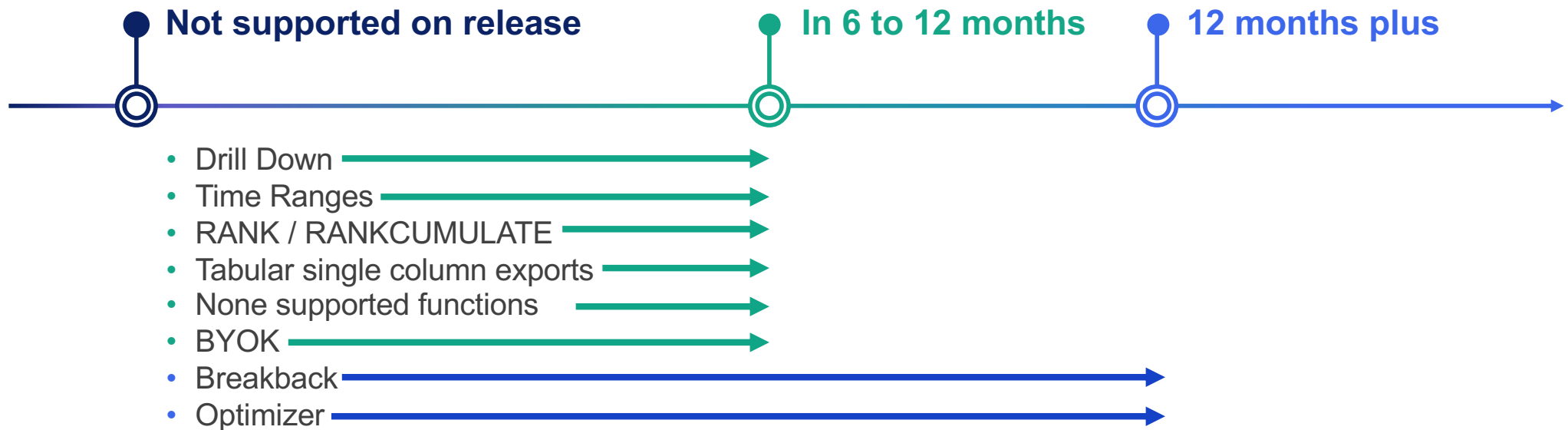


## Polaris Release Availability





## Functionality Support



Notes:

\* [Details for non-supported calculation functions here](#)

*Anaplan Confidential Information - Tentative Roadmap, all dates subject to change*

# How is Polaris Deployed?

## How is Polaris deployed.

- When created, a Workspace is set to be either 'Classic' or 'Polaris'
- Once created, the type of a workspace cannot be changed.
- To create a Polaris model, you simply create a new model in the Polaris workspace.
- It is possible to import/export between models in different types of workspace
- It is possible to copy Model Structure (via a Model Import) – provided that only features supported in Polaris are used in the source model

# Performance, Populated Cells, and Sparsity

## Performance

- The performance of a sparse engine is based on how the data is **stored** and organized and which results are **calculated**.
- Polaris performance is based on only storing combinations that hold data and only calculating combinations that may be relevant and generate results.
- To understand this, we need to explain
  - Sparsity & Density
  - Default Values
  - Calculation Complexity

## Naming and Terminology

- Sparsity / Density – is the ratio of populated cells to totally addressable cells. A Sparse model/module/line item is one where the vast majority of cells are not populated.
- Populated Cells – A populated cell is one that does not contain the Default Value.
- Default Value – The default value is currently fixed for each data type:
  - Numeric: Zero
  - Boolean: False
  - List Item: [Blank]
  - Date: [Blank]
  - Time Period: [Blank]
  - Text: "" [An empty string]

Notes:


\* Polaris has exactly the same semantics as the Classic engine when it comes to Numerics / Null.

## A Sparse Engine vs. a Dense Engine

In a Sparse Engine, only data values that are populated are stored. This is much more efficient for 'naturally sparse' data sets.

In a Dense Engine, like current Anaplan, memory is reserved for all possible data values in a coordinate space.

14 Cells with  
non-zero  
numbers stored  
in a Sparse  
Engine



	A	B	C	D	E	F
Bob		56	54		32	3
Tom			34	34		
Sam		34	43	77		
Eve	83				43	33
Iris	98				54	

All 30 Cells  
stored in a  
Dense Engine

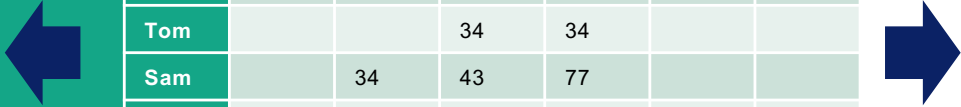


## The Memory required for each cell is also different

For a sparse engine, for every numeric Cell stored, approximately 24 bytes is required.

In a Dense Engine, like current Anaplan, each numeric cell requires 8 bytes.

14 Cells with  
Values x 24  
bytes = 336  
bytes in a  
Sparse Engine



	A	B	C	D	E	F
Bob		56	54		32	3
Tom			34	34		
Sam		34	43	77		
Eve	83				43	33
Iris	98				54	

30 cells x 8  
bytes = 240  
bytes in a  
dense engine

## One to One Calculations

- One-to-One formulas are where the calculation can be driven from one of the source line items in a way so that there will only ever be the same or fewer populated cells in the target line item than the sum of the input line items. A simple example is a multiplication (as we know that anything x zero is zero and therefore doesn't need to be calculated).

**Revenue**

=

**Units**

\*

**Price**

	Jan	Feb	Mar
Apple	£1		
Banana			£6
Cherry			
Date			
Elderberry		£2	

number **Revenue**[*Product, Time*]

	Jan	Feb	Mar
Apple	1		
Banana			3
Cherry			
Date			
Elderberry		4	

number **Units**[*Product, Time*]

Apple	£1
Banana	£2
Cherry	£1.50
Date	£3
Elderberry	£0.50

number **Price**[*Product*]

## 'One-To-Many' Calculations

- 'One-To-Many' Calculations are ones where the output can have more than the sum of populated cells in the inputs, but not every cell in the output has to be calculated. In a very simple example, if a Month dimensioned line item references a Quarter dimensioned line item, then the quarter value will be spread across the respective months. This will result in a greater number of populated cells in the target than the source.

**Units<sub>Month</sub>**

	Jan	Feb	Mar
Apple	4	4	4
Banana	5	5	5
Cherry			
Date	1	1	1
Elderberry			

number **Units<sub>Total</sub>**[*Product, Time*]

=

**Units<sub>Quarter</sub>**

	Q1
Apple	4
Banana	5
Cherry	
Date	1
Elderberry	

number **Units<sub>1</sub>**[*Product, Time*]

## 'All Cells' Calculations

- 'ALL Cells' formulas are where the calculation cannot be driven from any of the line item's populated cells. The only way to perform this type of calculation is to iterate over every cell in the target. A simple example is an Addition of a literal. In this case, every cell must be touched in order to calculate the result. Note that the result is not 100% Dense, but the calculation still has to iterate over every cell.

$$\mathbf{Units_{Total}} = \mathbf{Units_1} + 1$$

	Jan	Feb	Mar
Apple		1	1
Banana	1	1	4
Cherry	1		1
Date	-3	1	6
Elderberry	1	5	

number **Units<sub>Total</sub>**[*Product, Time*]

	Jan	Feb	Mar
Apple	-1		
Banana			3
Cherry		-1	
Date	-4		5
Elderberry		4	-1

number **Units<sub>1</sub>**[*Product, Time*]

+ 1

## Polaris Calculation Complexity

Formulas can be divided into 3 categories of Calculation Efficiency, depending on how they can be 'driven'

- **One-to-One** – This is where the output will be **As Sparse** as the sources. This type of calculation maintains sparsity, and is the most efficient category of calculation type.
- **One-to-Many** - This is where the output can be **More Dense** than the sources, but not every target cell can be populated. One-to-Many Calcs can be described using a '**Fan-Out Factor**' which describes the multiple of possible cells that can be populated.
- **ALL Cells** – This is where every cell has to be calculated **100% Dense**, and every cell could be populated. This is the least efficient type of calculation.

# Scale Limitations in Polaris

## Scale Limits in Polaris

There are two major scale constraints in Polaris:

1. The model must fit within a workspace. Just like Classic Anaplan, Polaris holds the whole model in memory within one running server. Polaris can make use of Hypermodels, but the model can't be larger than the workspace size.
2. The theoretical limit for a line item is  $2^{64}$  cells. This is approximately 18 quintillion cells (18 million trillion). The limit will vary depending on the shape of the dimensions, but the absolute maximum is  $2^{64}$  cells **PER LINE ITEM**.

The practical limit will always be less than this due to the representation of the dimension index. Nonetheless, a line item in Polaris can have trillions of addressable cells.

A graphic featuring two overlapping speech bubble outlines in white on a dark blue background. The left bubble contains a white letter 'Q' and the right bubble contains a white letter 'A'.

Q

A

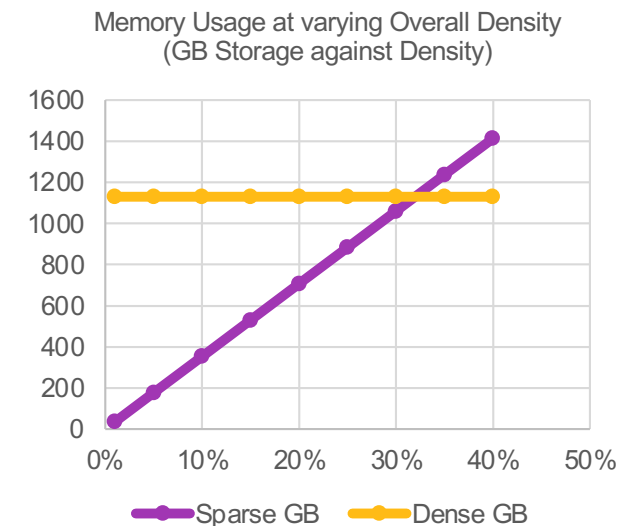


## Effect of Density on Memory Use for Sparse vs Dense Engines

In a Dense engine, each numeric value requires ~8 bytes for storage. As the whole hyperspace is stored – the total storage requirement does not change with density.

In a Sparse engine, each numeric value requires ~24 bytes for storage. Only the populated values are stored.

This means that for a given model, if the overall density is >33%, a dense engine is more memory efficient, and if <33%, a Sparse engine is more efficient.



Above chart showing memory usage for Inventory Planning example against varying overall density for sparse and dense engines.

## Aggregate Cells are usually not as Sparse as Primary Cells

Adding 1 level of rollup to each dimension in this example, moves the overall density from 47% to 70%

	Primary Cells	Aggregate Cells	Total Cells
Number of Cells	30	26	56
Populated	14	25	39
Density	47%	96%	70%

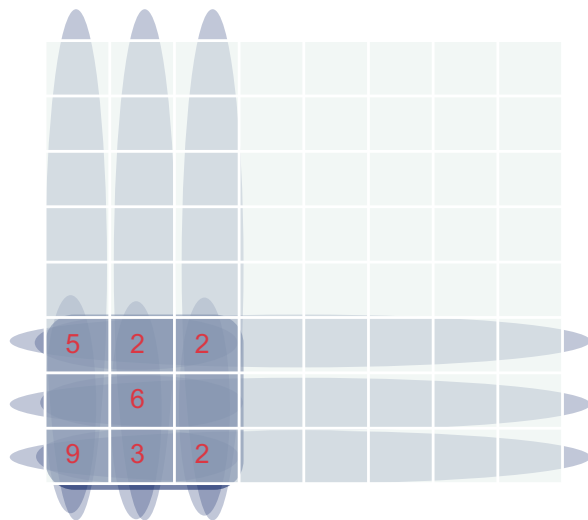
100% Dense	ABC	DEF
Men	144	69
Women	258	207

100% Dense	ABC	DEF
Bob	110	35
Tom	34	34
Sam	77	77
Eve	83	76
Iris	98	54

92% Dense	A	B	C	D	E	F
Men		56	88	34	32	3
Women	181	34	43	77	97	33

Primary Cells 47% dense		ABC			DEF	
		A	B	C	D	F
Men	Bob		56	54		32
	Tom			34	34	
Women	Sam		34	43	77	
	Eve	83				43
	Iris	98				54

The degree to which numbers of levels in hierarchies affect the overall density (and therefore size) of a dataspace is heavily dependent on the distribution of the data.



The same number of primary data cells can lead to very different numbers of populated roll-ups.

