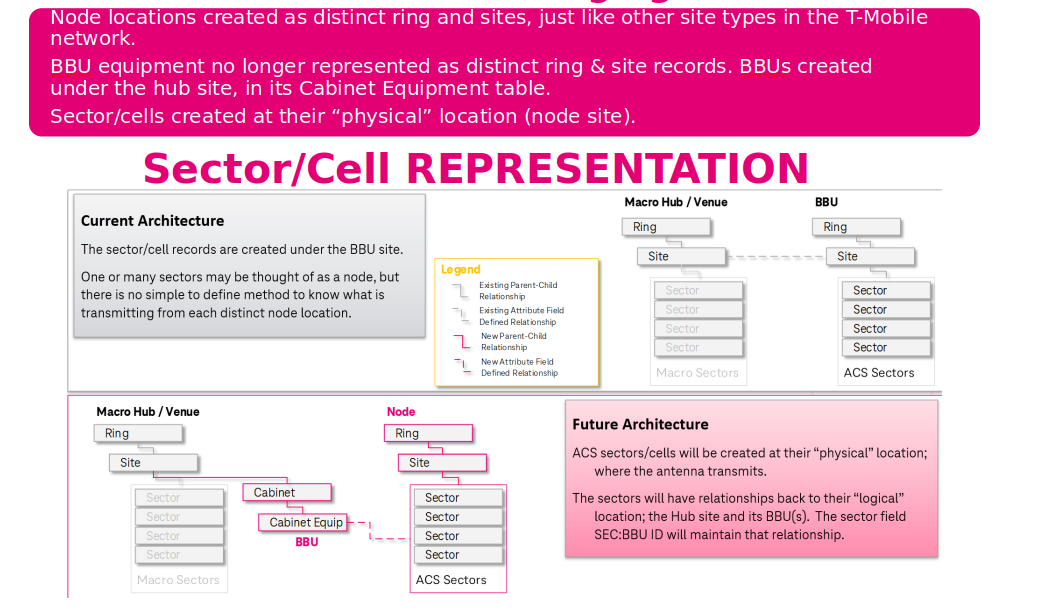
## ***SON Impact Analysis Node Restructure of Small Cells***

**Changes in Node Location Representation**



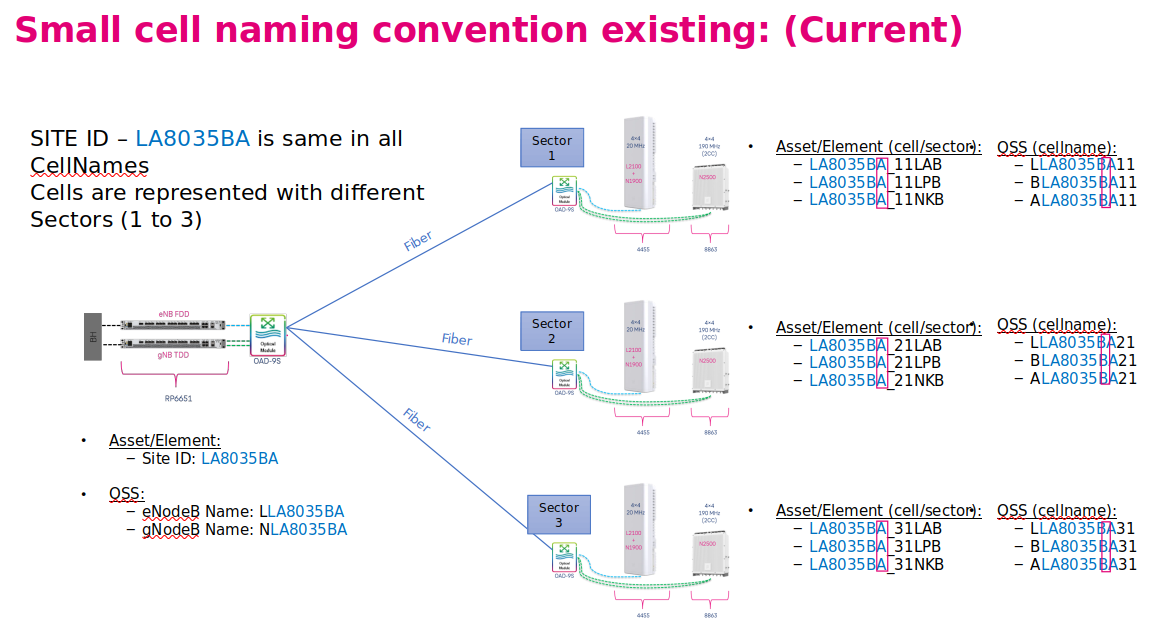
Node locations are now structured as distinct rings and sites, aligning with how other site types are organized in the T-Mobile network.  
BBU equipment will no longer be represented as separate ring and site records. Instead, BBUs will be included under the hub site within the "Cabinet Equipment" table.  
Sectors and cells will now be established at their actual physical location (node site).

The "SEC:BBU ID" field in the sector will continue to preserve the relationship with the hub site and its corresponding BBU.

**Sector/Cell Representation**

* **Current Architecture**: Sectors and cells are associated with the BBU site.
* **Future Architecture**: Sectors and cells will be directly created at their physical node location while maintaining their connection with the hub site and its BBUs.

**Existing Naming Convention**



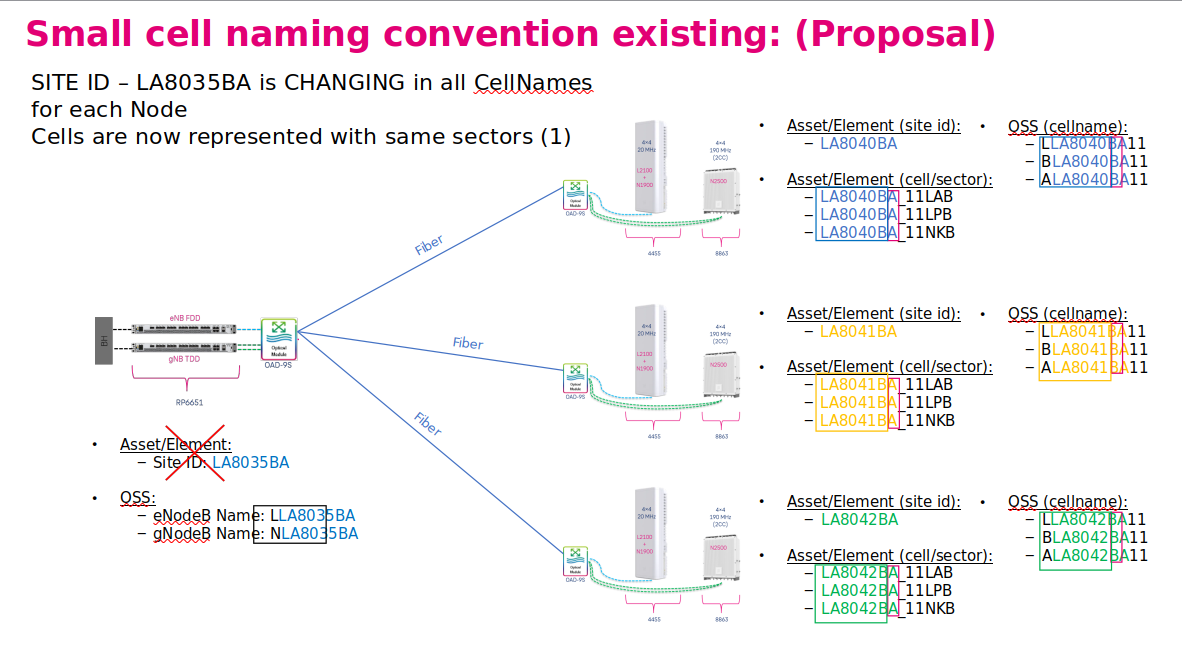
**Site ID:** A single Site ID is used for all cells within the site (e.g., LA8035BA).

**Cell Names:** Cell names are differentiated based on the sector they belong to.

* Example:
  + Sector 1: LA8035BA11LAB, LA8035BA11LPB, LA8035BA11NKB
  + Sector 2: LA8035BA21LAB, LA8035BA21LPB, LA8035BA21NKB
  + Sector 3: LA8035BA31LAB, LA8035BA31LPB, LA8035BA31NKB

**OSS Representation:** Cell names might be represented differently in the Operation Support System (OSS).

**Proposed Naming Convention**

****

**Site ID:** Each Node will have its own unique Site ID.

**Cell Names:** All cells within a Node will share the same sector number (e.g., Sector 1).

**Naming Format:**

* <Node Site ID><Sector Number><Cell Type>
* Example:
  + As you can see in the image, it lists unique site IDs for each node (e.g., LA8040BA, LA8041BA, LA8042BA).

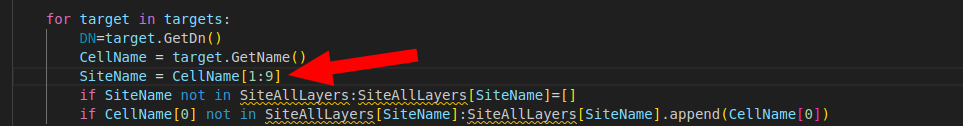
**OSS Representation:** The OSS representation will be updated to reflect the new naming convention.

**And to handle all the processes, we have created our own class called cell\_Details, details of which are given below**

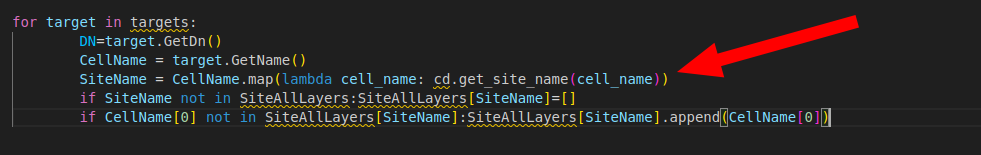
wherever string manipulation is performed, we need to replace it with the name of our custom class. For example, in the code, you will find various string manipulations like **cell\_name[1:8]** for site\_name, **cell\_name[1:9]** for site\_name, **cell\_name[1:-2]** for site\_name, **cell\_name[1:10]** for sector\_name, etc.

Our task is to identify these instances and replace them with data from our custom function. This approach will have several benefits, such as eliminating repetitive string manipulation and enabling our function to retrieve data from the cache, improving efficiency.

Let's understand with an example what the old code was and what the new code will be.



This is a part of the old code where we are retrieving the name using string manipulation



And this is the new code where we have created an object (**cd**) of our custom class and provided it with Site\_data or cell\_details information. Then, by using a function of this class, we are able to retrieve the information

**Let's now understand how the class is created and which functions are included under it.**

**Class Name: 'CellDetails'**

The CellDetails class provides essential information such as site\_name, sector\_name, and band\_name from site data (target\_data). It manages and caches key details related to cellular nodes, including node name, site name, sector, and band. By utilizing the enetsdk library, the class interacts with a database of cell details, offering methods to retrieve and store information. This approach improves efficiency by avoiding repeated data retrieval, and if the required data is not found in the cache, it will be fetched from enetsdk.

### **Class Breakdown**

1. **\_\_init\_\_(self, script\_data)**:
   * This is the initializer method where the class is instantiated.
   * **Parameters**:
     + script\_data: An object containing the necessary methods for retrieving data. It's used to interact with other systems (e.g., enetsdk and get\_cm\_reader).
   * **Attributes**:
     + cache: A dictionary to store cached cell details, avoiding repeated data retrieval.
     + script\_data: The script data object passed to interact with external data sources.
     + cdb: The enetsdk's GetCellDB() method, which helps retrieve cell-related data.
2. **run\_cache(self, target\_cells: list[object | str])**:
   * Caches cell details for a given list of target objects or cell names.
   * **Parameters**:
     + target\_cells: A list of either cell names (strings) or target objects.

**How It Works:**

1. Identifies whether target\_cells contains strings or objects with GetName attributes.
2. Resolves and retrieves cell details using get\_cell\_details() and stores them in the cache.
3. **\_get\_cached\_value(self, target\_cell: object | str, key: str)**:
   * Fetches a specific value from the cache for a given cell. If the value is not in the cache, it retrieves and updates the cache.
   * **Parameters**:
     + target\_cell: A cell name (string) or target object.
     + key: The specific detail to retrieve (e.g., nodeName, siteName, etc.).
   * **Returns**: The value for the specified key from the cache, or None if not found or on error.

**How It Works:**

1. Resolves the cell\_name based on the type of target\_cell.
2. If the cell\_name is not in the cache, fetches details and updates the cache.
3. Returns the requested value from the cache.
4. **get\_node\_name(self, target\_cell: object | str) -> str**:
   * Fetches the node name for a given cell.
   * **Parameters**:
     + target\_cell: A cell name (string) or target object.
   * **Returns**: The node name as a string.
5. **get\_site\_name(self, target\_cell: object | str) -> str**:
   * Fetches the site name for a given cell.
   * **Parameters**:
     + target\_cell: A cell name (string) or target object.
   * **Returns**: The site name as a string.
6. **get\_sector(self, target\_cell: object | str) -> str:**
   * Fetches the sector for a given cell.
   * **Parameters**:
     + target\_cell: A cell name (string) or target object.
   * **Returns**: The sector as a string.
7. **get\_band(self, target\_cell: object | str) -> str**:
   * Fetches the band for a given cell.
   * **Parameters**:
     + target\_cell: A cell name (string) or target object
   * **Returns**: The band as a string.

.

1. **get\_cell\_details(self, targets: list[object]) -> dict**:
   * This method processes a list of target objects and returns a dictionary containing detailed information about the cells, such as nodeName, siteName, sector, and band.
   * **Parameters**:
     + targets: A list of target objects (cells) to process.
   * **Returns**: A dictionary with cell names as keys and cell details (node name, site name, sector, band) as values.
   * **How It Works**:
     + First, it retrieves all site names for the given targets using get\_cm\_reader().read\_cells\_by\_uids().
     + Then, it processes each target:
       - Retrieves the cell name.
       - Determines the node name based on the technology (NR, LTE, 4G-NBIoT).
       - Retrieves the site name using the UID.
       - Derives the sector and band.
     + Finally, it stores all the retrieved data in a dictionary and returns it.

### **Conclusion**

### Refactoring and Optimization Overview for CmManager

This document outlines the recent changes and enhancements made to the CmManager module, focusing on replacing the old structure, improving performance, and introducing a new class for efficient data retrieval.

#### 1. Transition to a New Structure

* The earlier structure primarily relied on attributes such as site\_name and sector\_name.
* The objective of this update was to replace the old structure across all existing flows with a new, more reliable format.
* All occurrences of the old flow have been thoroughly reviewed and replaced with the new structure, ensuring consistency across the codebase.

#### 2. Enhancing Efficiency and Reliability

* Challenge Identified:
  + The previous process required fetching data directly from the SDK API for every request. This approach was not only slow but also less efficient.
* Solution Implemented:
  + To address this, a caching mechanism has been introduced to minimize repetitive API calls.
  + This enhancement ensures faster response times and reduces the dependency on continuous SDK API requests.

#### 3. Introduction of the CellDetails Class

* To further optimize data retrieval, a new class named CellDetails has been implemented.
* Purpose of the Class:
  + The CellDetails class acts as a bridge between the new structure and efficient data caching.
  + It retrieves data in the new format while storing frequently accessed data in a cache for faster subsequent access.
* Key Benefits:
  + Eliminates repeated API calls for identical data.
  + Accelerates the data retrieval process by leveraging in-memory caching.
  + Ensures a seamless transition to the new structure while maintaining performance.